Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

DOCTOR OF SCIENCES

Development and Study of Optimization Algorithms for Medical Image Registration Problems

Buhendwa Nyenyezi, Justin

Award date:
2018

Awarding institution:
University of Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

UNIVERSITÉ DE NAMUR

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUE

Development and Study of Optimization Algorithms for Medical Image Registration Problems

Thèse présentée par
Justin Buhendwa Nyenyezi
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury:

Prof. Lieven De Lathauwer
Prof. Anne Lemaître (Président du jury)
Dr. Hubert Meurisse
Prof. Daniel Ruiz
Prof. Annick Sartenaer (Promotrice)

2018

Abstract

In medical image analysis, image registration aims at analyzing several images acquired at different times or by different devices. More precisely, it determines the suitable spatial transformation that allows these images to be aligned in a common spatial domain. The common issues that are known for the deformable 3D image registration problem include the computing time that, especially for non-parametric transformations, may be quite troublesome for some clinical applications. This thesis is concerned with the analysis of numerical algorithms designed to solve efficiently the medical image registration problem with a focus on the use of preconditioners to speedup iterative linear system solvers. Our purpose is two-fold. First, we propose an extension in the use of an existing package, FAIR from Jan Modersitzki, by allowing the user to choose polynomial preconditioners and/or to choose large deformations. Second, if applicable, we propose to use a compressed representation of data with a given accuracy ϵ using Tensor-Train format to solve efficiently the linear systems. Within this tensor format, a low-rank preconditioner built with spectral information is used to speedup and stabilize the system solver.

Résumé

En imagerie médicale, le recalage des images a pour objectif d'analyser plusieurs images acquises à différents moments ou par différentes techniques. Plus précisément, le recalage d'images médicales permet de déterminer la meilleure transformation spatiale qui permet de passer d'une image à l'autre. Une telle transformation facilite l'analyse de ces images car elle permet que ces images soient alignées dans un même domaine spatial pour être comparées. Cette thèse propose une analyse de quelques algorithmes numériques utilisés pour résoudre efficacement le problème du recalage non rigide d'images médicales 3D pour des transformations non paramétriques. Cependant, il est établi que, de manière générale, ces algorithmes sont coûteux en temps de calcul et en mémoire de stockage. Ceci peut se révéler problématique pour certaines applications thérapeutiques. Dans le but d'accélérer ces algorithmes, cette thèse s'est focalisé sur l'étude du préconditionnement des systèmes linéaires résolus itérativement dans la phase d'optimisation du recalage. Cette thèse propose une extension du package FAIR de Jan Modersitzki. D'une part, elle permet à l'utilisateur de faire appel à des préconditioneurs polynomiaux, en plus des preconditionneurs déjà existant dans le package. D'autre part, si faisable, elle lui propose le choix d'utiliser des solveurs en un format tensoriel. Ce dernier, exploitant la structure des données, permet une compression efficiente et une troncature à un seuil de précision ϵ donné qui fait gagner du temps et de la mémoire de stockage.

Remerciements

Mon âme loue l'Éternel Dieu!

Je suis infiniment reconnaissant envers l'université de Namur pour le soutien financier à travers la bourse CERUNA dont j'ai été bénéficiaire.

Je remercie particulièrement ma promotrice Annick Sartenaer qui a rendu possible ce travail. Non seulement elle a accepté de le diriger, malgré ses diverses tâches et responsabilités, mais en plus elle n'a ménagé aucun effort pour rendre possible et viable mon séjour en Belgique. La confiance qu'elle m'a accordée a permis que je développe le travail que j'avais envie de faire. En même temps, sa rigueur scientifique qui se manifestait dans les nombreuses questions et la remise en question des moindres détails non argumentés, a fait que j'acquière auprès d'elle une véritable formation de mathématicien avec une bonne qualité. Elle est pour moi un véritable modèle scientifique. Sans sa relecture méticuleuse, ce travail n'aurait pas sa qualité actuelle. Annick, je n'aurai jamais de mots pour t'exprimer ma gratitude pour ta bienveillance, ton attention, ton amitié et qu'en sais-je encore! Infiniment merci!

Merci sincèrement aux membres de mon jury, le prof. Daniel Ruiz, le prof. Lieven De Latauwer, la prof. Anne Lemaitre (président du jury) et monsieur Hubert Meurisse pour leurs remarques pertinentes, leurs conseils constructifs et leurs différentes suggestions lors de la défense privée.

Un très grand merci à mon papa Pascal Muhindo Nyenyezi et à ma maman Ghislaine M.ciroha pour leur amour sincère. Que mon frère Vital Rhushenge Nyenyezi et mes sœurs Justine Ashuza, Christine Musimwa et Marthe Alola se sentent remerciés pour leur soutien moral. Vous avez toujours été là pour moi.

Je remercie l'ISP/Bukavu qui m'a laissé libre pendant ma formation. En particulier je pense aux collègues des départements de math-physique et de physique-technologie. Restons unis pour relever les défis de l'éducation aux sciences dans notre région. Merci à l'ARES-CUD (de Belgique) qui m'avait octroyé une bourse pour un master complémentaire.

Je remercie Jean Tshimanga Ilunga, avec qui, depuis le début de l'aventure on a eu des discussions fructueuses et a conduit mes premiers pas dans le monde de l'algèbre linéaire numérique. Merci papa Jean et ta famille qui m'a accueilli plus d'une fois à Toulouse.

Durant ma recherche, j'ai fait des rencontres et j'ai eu des conseils constructifs sans lesquels je n'aurais pas fini ce travail. L'accompagnement du prof. Renaud Lambiotte et de Hubert Meurisse m'a été d'une utilité considérable mais aussi les discussions avec Damien De Nizza. Qu'ils se sentent remerciés. Mes sentiments de remerciement vont aussi à madame Daniella di Seraphina avec son équipe en Italie qui m'a mis sur des pistes intéressantes. Merci aux amis(es) qui m'ont beaucoup aidé quand j'arrivais au département. Je pense à Phillipe Sampaio, à Charlotte Tanier, à Anne-sophie Crélot, à Delphine Nicolay, Cédric, surtout Éric Cornelis, Pascale Hermans et bien d'autres.

Que tous les membres du département se sentent remerciés ici. Ils ont tous été bienveillant à mon égard et j'avais toujours de l'aide quand je le sollicitais.

J'ai eu cours avec certains, de belles discussions avec d'autres et d'autres encore ont été présents financièrement. Par exemple, Anne Lemaitre quand elle était chef de département, n'a pas hésité à me faciliter un aller au Congo mais aussi à suivre quelques leçons d'anglais. Quand je n'avais plus de bourse, le bureau du département, dirigé par Joseph Winkin, m'a trouvé un financement, merci. Merci père Marcel Rémon, merci André Hardy, merci Martine, merci Alice, merci Marvyn, merci Isabelle, oups! je ne saurai pas citer tous! Merci à tous.

Merci à la communauté du Temple Protestant Évangélique de Namur et pour la représenter je cite les pasteurs Laurent Sulpizio et Bernard Laverdure. Merci à la communauté de pères Barnabites et particulièrement le père Étienne Ntale et le père Ferdinand Mushagalusa tous à La Louvière.

Merci à la famille Mac Mugumaoderha Cubaka. Quoi dire pour cette famille, vous avez été tout pour moi. Je me sents chez moi vraiment. Merci Mac, bisous Lune, Lisa, Linda, Lucie, Lorie et trois têtes à Robin. Vous êtes vraiment ma famille. Merci à la famille Jean Hamuli, la famille Aimé Kachungunu, la famille Bébel Mitima, la famille Élie Zihindula, la famille Belma Malanda, la famille Jean de Dieu Kahirika, la famille François Mulangaliro, la famille Machumu, la famille Bujiriri et Esther à Paris et les autres membres de la communauté Kivutienne de Belgique. Merci à l'asbl AFESDI avec maman Joséphine Bashengezi.

Que le professeur Daniel Tuyttens (Umons) avec qui on a écrit un premier projet de thèse se sente remercié. Également, merci à la professeur Sabine Limbourg (Uliège) pour sa recommandation. En amont de ma thèse, j'ai eu un soutien considérable du grand frère et ami Beauty Kazige Mushamuka et puis du couple Kaski à Bruxelles. À eux je dis, grand merci.

Sans les citer tous, je pense à de nombreux(ses) amis(es) qui m'ont été très utiles d'une manière ou d'une autre. Fabrice Muvundja, Philippine Sakina, Juliette Passy, Gaston, Christian Nazili, Nathalie Masudi, Souzy Mbaka, Godelive Batano, Constance De combrugghe, Christel Lamère Ngnambi, Victoria Brandemann, Jacques Galangwa, Joseph bavurha, Séraphin Bireke, Isaac Makelele, Douce Makelele, Tessa Bwandinga, Christian Mugisho, Félicien Rafiki, Doux, Joelle Baleke, Quentin, Dorine, Jearice, Joriane, Noémie, Darius Makindu, Didier Kumwimba, Dieudonné Ecike Ewanga, Pascal Shogolo, etc.

Que ceux qui ne sont pas cité ici ne se sentent oubliés, je leur dis sincèrement merci, certains pour leur gestes et d'autres pour leurs mots.

Contents

Introduction

In medical image analysis, it is common to analyse several images acquired at different times or by different devices. This analysis may need a spatial transformation that allows these images to be aligned in a common spatial domain and correspondences to be established between them. This process is called *Image registration*. Let us consider one of the images as *the reference image*. The registration process aims to determine the *suitable spatial transformation* that deforms each of the other images, such that it becomes *as similar as possible* to the reference image.

Although this problem seems relatively easy to address, its numerical resolution faces significant technical and practical issues [1]. Firstly, this problem is, in general, ill-posed according to Hadamard. This means that the existence and uniqueness of the solution are not guaranteed, or the behaviour of the solution may not depend continuously on the initial conditions. Secondly, the choice of the *dissimilarity measure*, of *the deformation model* and of *the optimization method* entails a compromise between the efficiency of the algorithm and the richness of the description of the solution.

This compromise requires a good undestanding of the acquisition process and the specific application. Furthermore, even when the problem is finally well-posed, it is often *ill-conditioned*. Roughly speaking, this tells us that small changes in the data induce significant changes in the results. To tackle these issues, many registration algorithms have been proposed. They are essentially distinguished by the way of choosing the dissimilarity measure, the transformation space and the optimization method. A general review of these algorithms is available in the surveys: [1], [2], [3] and references therein.

These days, rigid registration methods that use rotations, translations and affine transformations are well understood. At the same time, almost all the algorithms that use non-rigid transformations and that deform local regions within the image, encounter many challenges. These challenges are related to topology preservation, ill-posedness, ill-conditioning, inverse consistency, computing time and storage memory demands. These issues are even more critical

for 3D/4D high-resolution images.

For these reasons, many algorithms have been developed within the *Demons algorithms* proposed by J-P Thirion [4]. Demons algorithms are popular and widely used in professional contexts thanks to their linear complexity. However, for high-resolution and/or high-dimensional images (3D or even 4D), this linear complexity may be unsatisfactory for clinical or therapeutic purposes. One way of accelerating these algorithms is the use of super computers that can do several computations in parallel and in a record time. But the cost of such machines remains high and unaffordable for many hospitals. Currently, reseachers are turning to other, less costly techniques such as the use of certain computer architectures (for example, the CUDA from NVIDIA or Radeon HD from AMD) [5] and through development of more efficient mathematical algorithms. It is with this last perspective in mind that this research is oriented.

Scope and main contributions

This work aims to tackle the non-rigid and nonparametric registration problem for high-resolution 3D medical images, through the development of numerical optimization algorithms which offer a good compromise between complexity and speed. To address this large-scale problem, we may refer to methods that allow to reduce the storage and the cost of numerical operations. Let us assume a d-dimensional array ($d \geq 3$) with n entries in each direction. The data size of such an array is $N = n^d$. For n sufficiently large (e.g, $n = 10^3$), even linear complexity $\mathcal{O}(N) = \mathcal{O}(n^d)$ might be far beyond any computer capacity [6, p.vii]. For this purpose, the task is to speed up the process with techniques that offer possibilities of replacing a linear complexity $\mathcal{O}(N)$ with a quasi logarithmic complexity $\mathcal{O}(d\log(n))$. This supposes the use of a compressed representation of data with a given accuracy ε. Indeed, over the past few years in the field of scientific computing, compression techniques for high-dimensional data using suitable data sparse representation have been developed [6]. These representations, generally known as *numerical tensor representations* or *numerical tensor formats*, are inspired by hierarchical matrix techniques, variables separability and low-rank approximations. They allow the handling and solving, in an efficient way, of some numerical smooth problems with high-dimensional data (see [6], [7], [8] and [9]). According to [6], under suitable conditions, some of these representations (hierarchical tensor format and Tensor-Train format) are stable and hopefully allow a reduced complexity from $\mathcal{O}(n^d)$ to $\mathcal{O}(d\log(n))$. For numerical resolution of large optimization problems, such as systems encountered in 3D image registration, iterative algorithms such as the Preconditioned Conjugate Gradient (PCG), the Minimal Residual method (MINRES) or the Generalized Minimal Residual methods (GMRES), can be adapted to integrate these numerical tensor approaches, see [9]. It is in this perspective that this research project has been conducted. The two main contributions of this work may be summarized as follows.

Design of appropriate preconditioners

In this work, we provide an analysis of certain preconditioners used to solve large linear systems with Symmetric Positive Definite matrices arising in medical image registration (see Chapter 4). We highlight why and how the Tchebychev polynomial preconditioner is well suited for such problems (see Section 4.4 and Section 4.4.3). We further illustrate this by numerical experiments in Section 4.6.

Extension of the FAIR package

The Flexible Algorithms for Image Registration (FAIR) package was proposed by Jan Modersitzki [10]. Although this package provides general algorithms for any image (in dimension 2D and 3D), it focuses on medical images in 2D. FAIR package uses the PCG-solver with Jacobi, Gauss-Seidel and incomplete Cholesky preconditioning techniques to speed up and stabilize the successive linear systems arising in the registration process. In this thesis, we propose an extension of this package by providing an integrated and flexible algorithm, FA4DMIR in Section 5.4 that allows to use, in addition to those implemented in FAIR, polynomial preconditioners. In addition, this algorithm allows a PCG solver in Tensor-Train format with low-rank preconditioners (see Section 5.3.5). The goal is to address larger images, in particular, 3D medical images for which there may be a constraint on computing time and thus a compromise between speed and precision. Numerical experiments in Chapter 6 illustrate effectiveness of this algorithm on 3D medical images compared to those implemented in FAIR.

Organisation of the thesis

This work is subdivided as follows:

Chapter 1 explains different steps of the registration process. It provides an overview of the deformable medical image registration problem, from images acquisition to the registration results, but does not address the interpretation of the results.

Chapter 2 basically recalls numerical optimization methods, while the focus is on line-search strategy and the Conjugate Gradient methods that are used in the final algorithm.

Chapter 3 presents the test environment used to compare certain linear system solvers and certain registration algorithms.

Chapter 4 presents a discussion on some matrix preconditioning techniques for large and sparse linear systems. Some preconditioners are explained and compared. In the final algorithm, the user is allowed to call one of the proposed preconditioners, depending on their context and need.

Chapter 5 addresses the numerical tensor formats, where the focus is on low-rank approximations and linear system solvers in Tensor-Train format. The

last section of this chapter presents the proposed algorithm for Medical Image Registration Problems that allows to use some preconditioner studied in previous chapter. This include the possibility of solving the linear system in tensor format.

Chapter 6 presents some numerical experiments where three algorithms are compared via the perfomance profile tool and some simulations on certain images are presented. A short conclusion ends the work.

Chapter 1

Deformable medical image registration

In this chapter, imaging techniques and deformable image registration methods are presented. The intention is to enable the reader to understand concepts and strategies behind the main registration algorithms while underlying these concepts and their mathematical formulations. Medical image registration has a wide range of potential applications [11, p.12]. These applications include combining information from multiple imaging modalities, like for example, relating functional information to structural information within images.

Image registration is also used for monitoring changes in shape, size, or intensities over time. Another application consists of relating preoperative images and surgical plans to the physical reality of the patient in the operating room during image guided surgery (or in the treatment suite during radiotherapy). Finally, registration is used in relating an individual structural image to a standardized atlas.

1.1 Image processing: an overview

The dramatic increase in availability, diversity and resolution of medical image devices over the last few years calls for modern image processing techniques. This challenge has been addressed in the field of *artificial vision* via mathematical models and algorithms whose simulations may be regarded as the end products. However, it should be noted that medical images suffer from one or more of the following imperfections: low resolution (in the spatial and spectral domains), high level of noise, low contrast, geometric deformations or the presence of imaging artefacts, etc. [12, p.7].

Most of these imperfections can be inherent to the imaging modality or the results of a deliberate trade-off during acquisition. For example, X-ray images offer low contrast for soft tissues, ultrasound produces very noisy images and metallic implants will cause imaging artefacts in magnetic resonance images (MRI images). Also, to get finer spatial sampling, a longer acquisition time may be needed, while this increases the probability of patient movement and thus blurring. Here are five key tasks that have been performed to support the eye-brain system of the medical practitioners, for image analysis.

Segmentation

Image segmentation is a process whereby a structured visual representation is created from an unstructured or less structured image. In current formulation, the image segmentation aims to partition an image into homogeneous regions that are semantically meaningful and easy to identify [13].

Image Smoothing

Smoothing is used in order to reduce noise and useless details within the image, so as to simplify its analysis. This action of simplifying an image has to preserve all important information and avoid too much distortion [14].

Registration

Registration is the process whose aim is to find a spatial transformation that aligns multiple data sets in a common spatial space each other. This is the topic of this thesis.

Visualization

Visualization of medical images includes the technological environment in which image-guided procedures can be displayed. It allows the determination and highlighting of information related to anatomical and functional properties of tissues, often affected by disease [12] (see also [15]). In medical image processing, one refers to an anatomical reference providing orthogonal plans as reference.

In Figure 1.1 from [16] the configuration of these plans is shown. Three orthogonal plans are distinguished: axial (transverse) plan, coronal (frontal) plan and sagital plan.

Simulation

Medical images play another important role due to simulations. They are used to prepare, control and validate a specific therapy. Today's simulations are

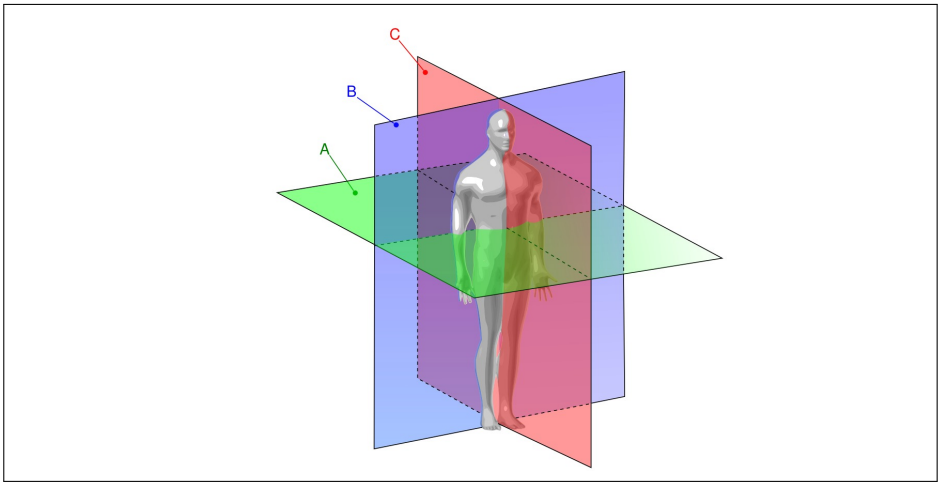


Figure 1.1: Anatomical reference plans. Axial plan (A), coronal plan (B) and sagittal plan (C) [16].

often used as an intermediate step between diagnosis and therapy itself. For example, a virtual patient or organ may be built to simulate results of a given therapy, an evolution of an artificial pathology or to control surgery [17].

This thesis has a focus on the registration problem. However, it is common to integrate all these technologies into complete and coherent image-guided therapy delivery systems. In such systems, the ultimate user is a human who judges the utility of the procedure and who tunes the parameters [12].

1.2 Introduction to medical imaging

According to [12], medical images are acquired by very sophisticated techniques that cross all biological scales beyond the visible photographs and microscopic images of the last century.
The evolution of medical imaging technology allows the organization of observations of biophysical phenomena and an increased ability to apply new processing techniques. Moreover, today's medical imaging techniques allow multiple medical data to be combined into complex mathematical models of physiological functions.

These medical imaging provide information on diverse physical phenomena, such as the time variation of haemoglobin deoxygenation during neuronal metabolism, or the diffusion of water molecules through and within tissues. Medical image analysis is mainly used in the therapeutic process: Image-Guided Therapy (IGT) and Image-Guided Surgery (IGS) for localization, tar-

geting, monitoring, and control. For example, a doctor can non-invasively monitor the healing of a damaged tissue or the growth of a brain tumor, and determine an appropriate medical response.

Many different imaging techniques are used for clinical purposes. They are based on different physical principles that can be more or less suited to a specific pathology (see [18]). Two families of modalities are distinguished: structural and functional modalities. These modalities are sometimes considered as complementary when they offer different insights into the same underlying reality. For example, one may fuse a functional and a structural image to localize a tumor with precision.

Figure 1.2a illustrates, in its first row, functional images (coronal, sagital and axial slices), where a tumor is in activity. To localize the tumor with precision, structural images are needed (they are visualized in the second row). Finally, a registration is needed to fuse these images and localize the tumor. This is visualized at the third row (images from [19]). Further, hands images from FAIR of Jan Modersitzki [10] illustrate the fusion of monomodal registration results. The top image in 1.2b is a fixed image (with red landmarks), the middle image is an image (moving image with green landmarks) to be deformed and the bottom image is the fusion of these images after registration.

Structural or anatomical modalities

The anatomical modality provides informations about the anatomical structure. This modality includes, but is not limited to:

Radiography imaging (from X-rays)

X-ray imaging is the oldest medical imaging [1]. For this, the patient is positioned between an X-ray tube that produces a set of electromagnetic waves (X-rays of high-energy photons) and an X-ray detector. On the X-ray detector, the X-rays are reconstructed within a degree of attenuation depending on the properties of each scanned point of the patient's body. These different attenuations allow different organs of the patient to be distinguished [12].

Ultrasound (US) imaging

Ultrasounds are able to distinguish subtle variations among soft and fluid-filled tissues. With this technique, high frequency sound waves are sent by a

1. From 1895, X-rays were discovered by Roentgen, but at that time it was not understood that these X-rays are ionizing and that high doses are dangerous [12].

transmitter into the body. There, they produce different patterns of echoes, bouncing off the different organ tissues. Acquired by a receiver, these echoes are forwarded to a computer which translates them into an image. As opposed to X-rays, ultrasound techniques are not dangerous for organ tissues with ionizing radiation. However, they produce very noisy images and require preprocessing to distinguish very small features such as cysts in breast-like imagery.

Computed Tomography (CT) imaging

CT images are 3D reconstruction of 2D X-ray radiography, reconstructed by computers from 2D projections using the Radon transform [2]. These 2D radiographs are acquired by rotating the X-ray tube around the body of the patient. Between bones and soft tissues, CT images offer high contrast while they offer low contrast between different soft tissues. To improve the image quality, a chemical solution (opaque to X-rays) can be injected into the patient to increase the contrast between tissues.

The difference between CT scan and X-ray imaging is that waves produced by X-ray are emitted in one direction while those produced by CT-scan imaging are emitted in all possible directions. It is important to note that as CT is based on multiple radiographs, the effects of ionizing radiation should be considered. However, with modern devices it is claimed that the dose is sufficiently low that there should be no major health risk issue.

Magnetic resonance imaging (MRI)

In this type of imaging, hydrogen nuclei of water are magnetically excited in the body and the image reconstruction relies on their relaxation properties.

Following [12], the *relaxation process* is briefly described as follows: the patient is exposed to a burst of radio-frequency energy which elevates the energy state of the nuclei in the presence of a magnetic field. Then, molecules shed this energy into their surroundings while they undergo their normal microscopic tumbling. Different tissues provide different relaxation rates that allow to construct the image. MRI techniques are said to be harmless to patients because they use strong magnetic fields and non ionizing radiation in the radio-frequency range. Tissue contrasts within MRI techniques are better than those from X-rays and provide higher-quality images, especially in brain and spinal cord scans.

2. Let $f(X) = f(x, y)$ be a compactly supported continuous function in $\mathbb{R}^2$. The Radon transform Rf is a function defined on the space of straight lines L in $\mathbb{R}^2$ by the line integral on each such line: $Rf(L) = \int_L f(X)|dX|$ [20].

Functional modalities

This modality provides insight into the metabolism of the underlying organs. They include nuclear medecine modalities (Scintigraphy, Single Photon Emission Computed Tomography (SPECT), Positon Emission Tomography (PET)), Functional Magnetic Resonance Imaging (fMRI) and others related.

Positon Emission Tomography (PET)

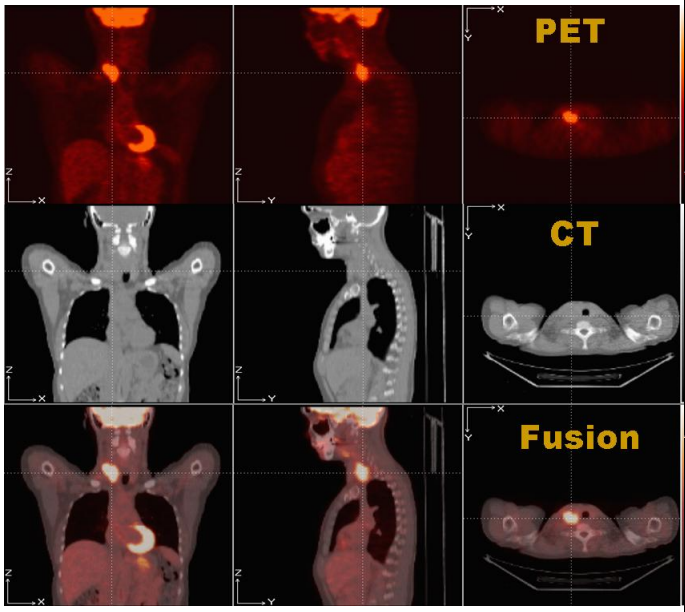
Two types of 3D nuclear medicine imaging studies include the Single Photon Emission Computed Tomography (SPECT) and the Positron Emission Tomography (PET) [26, p.15]. In general, the SPECT studies use radiotracers that emits photons while decaying and the PET studies use radioactive isotopes that decay by positron (also called positons, they are anti-electrons) emission. In fact, radioactive isotopes are injected into the patient that emits positons. The collision between positons and electrons produces pairs of gamma ray photons moving in different directions but having the same energy. The origin of the photon pair can then be determined from their positions delay. Thus, if one uses radioisotopes that have different properties for different tissues, this modality allows pathologies to be visualized at the much finer molecular level. However, the radiation dose in PET imaging is similar to a CT scan. Image resolution may be poor and need preprocessing, see Figure 1.3d (see [21]).

Functional Magnetic Resonance Imaging (fMRI)

This functional modality is a refinement of the structural MRI that measures temporal variations and diffusion of water molecules in an anisotropic environment. For more details see [12, 22] and references therein.

The most used image modalities are illustrated in Figure 1.3, where samples of different modalities are shown. Although CT, US and MRI modalities are all used for structural images, one can see that CT scan (see (1.3b)) images are better for less soft tissues (for example, bone injuries, chest problems). On the other hand, MRI-images (see (1.3c)) are well suited for soft tissues (for example: pain in muscles, tendons or ligament injuries and brain tumors). However, MRI imaging is expensive and time consuming. For this reason, US imaging (see (1.3a) is more often used. For functional images, PET-images (see (1.3d)) are less expensive but their resolution is not satisfactory. Many alternatives have been proposed within fMRI (see 1.3e) and mixed imaging such as PET-CT imaging (see 1.3f).

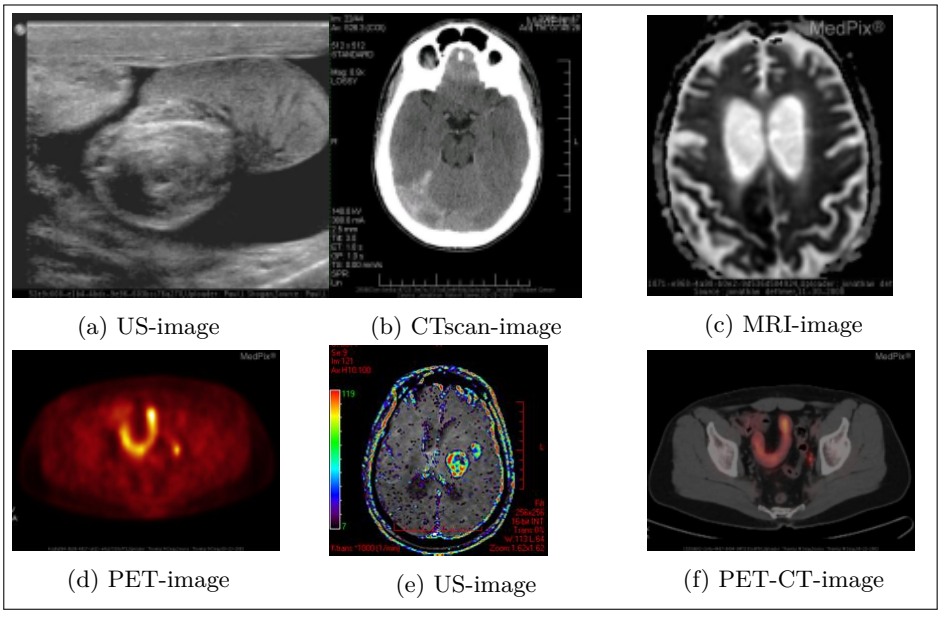
Properties of main imaging modalities are summarized in Table 1.1. In this table, we present the main characteristics that can influence either the interpretation of registration results or certain behaviours in numerical treatment.



(a) Functional images at the top row, and structural images at the middle row are fused using registration. The bottom row shows results of the fusion and allow to localize the tumor with precision [19].

(b) Monomodal MRI hands fusion after registration with landmarks [10].

Figure 1.2: Multimodal registration (images for tumor are multimodal: CT and PET) and monomodal registration (hands images are monodal: all MRI)



(a) US-image (b) CTscan-image (c) MRI-image

(d) PET-image (e) US-image (f) PET-CT-image

Figure 1.3: Illustration of most used medical image modalities. Images from [23].

	CT	MRI	PET	SPECT	US
Main characteristics	Scan body organs using X-rays.	Produce "slices" that represents human body through applying magnetic signals	Use nuclear imaging techniques and use tracers for disease diagnosis	Non invasive based technique where cross-sectional images radiotracer within human body are structured	Produce quantitative & qualitative diagnostic information via sound waves based techniques that possesses high temporal frequencies
Advantages	Wide field of view Detection of even subtle differences between body tissues Eliminates superposition of images from overlapping structures High spatial resolution High penetration depth Clinical translation.	High resolution, Able to show anatomical details Does not use ionizing radiation No short term effects are observed	Used to distinguish between benign and malignant tumors in a single imaging Can image biochemical and physiological phenomena High sensitivity High penetration depth	Images free of background Confirm neurodegerative diseases High sensitivity(but lower that PET) High penetration depth	High spatial resolution Low cost Safety profile Non invasive (no noodles or injections) widely available No radiation Easy to use
Disadvantages	Limited sensitivity Radiation High dose per examination Cost Non specificity for tissues Poor soft tissue contrast	Strong magnetic field disturb Can not be used in patients with metallic devices, such as pacemakers Low throughput Cost	Limited spatial resolution Radiation High costs Most expensive technique Motion artifacts are the serious problems Lower resolution compared to CT and MRI Interpretation is very challeging	Blurring effects are produced Attenuation compensation is not possible due to multiple scattering of electrons Fails to predict neuropsychological deficits Limited spatial resolution Radiation	Operator dependent Imaging limited to vascular compartment Difficult image pf bone and lungs Limited resolution Attenuation can reduce the image's resolution Reflected very strongly on passing from tissue to gas or vice versa
Contrast	High	High			
Cost	Intermediate	Intermdiate	High	High	Low
Radiation source and type	X-rays(ionizing)	Electric and Magnetic Fields (non ionizing)	Positron(ionizing)	photons(ionizing)	Sound waves (non ionnizing)

Table 1.1: Comparison between the main medical imaging techniques. Inspired by [18, p.111-112, Table 1]

1.3 Medical Image Registration Problem (MIRP)

In this section, concepts of image, grid and image registration are presented. Spline cubic interpolation and three main key steps of the registration process are highlighted: *dissimilarity measure*, *transformation models* and the *optimization method*. The optimization step will be the focus in this thesis. This section is mainly inspired by publications of Jan Modersitzki within [10, 24].

1.3.1 Image and grids

Given a spatial domain $\Omega \subset \mathbb{R}^d$, an image is considered as a mapping which assigns to every spatial point x belonging to Ω, a gray value $b(x) \in \mathbb{R}$. This gray value, considered as the amount of energy from a light wave at the position x, is called the image intensity at x or the voxel value (pixel value in 2D). The image intensity being a non negative real value, the image is formally defined in the following way [24].

Definition 1.1 (*Image*)

Given $d \in \mathbb{N}$, a function $b : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ is a d-dimensional image if

1. *b is compactely supported,*
2. *$0 \leq b(x) < \infty$, $\forall\, x \in \mathbb{R}^d$,*
3. *for $k > 0$, $\int_{\mathbb{R}^d} b(x)^k dx$ is finite.*

The set of d-dimensional images is denoted by

$$Img_{\Omega}(d) = \{b : \Omega \subset \mathbb{R}^d \to \mathbb{R},\ b \text{ is a } d\text{-dimensional image}\}. \tag{1.1}$$

The spatial dimension d of the image is generally 2 or 3. The set $Img(d)$ of all d-dimensional images is a differentiable manifold [24].

In practice, an image is determined through the intensity values at each spatial point $x \in \mathbb{R}^d$. As the space $\mathbb{R}^d$ is infinite, the stored images (these are discrete images) are samples of these continuous functions at some specific points.

In the following, a discrete image will be denoted by $dataI$ and a continuous image obtained by interpolation will be denoted $I(x)$ or simply I. Here, only cubic spline interpolation is used because it is, at least, twice differentiable and thus useful for fast continuous optimization methods. The interested reader is referred to [10] and references therein for other interpolation techniques.

Numbering, cells and Grids

Consider $I : \Omega \subset \mathbb{R}^d \longrightarrow \mathbb{R}$, a d-dimensional image on a d-dimensional domain $\Omega = \times_{l=1}^{d}[\omega_{2l-1}, \omega_{2l}]$. It is assumed that the support [3] of I is included in the domain Ω, and we shall denote the boundary of Ω by $\partial\Omega$. Image intensities are sampled on some specific points of the domain. The set of all these points is called a grid as defined in the following definition.

Definition 1.2 (*Grid points, Grid, Grid vector*)

Let $d \in \mathbb{N}$, $n^{(1)}, \ldots, n^{(d)} \in \mathbb{N}$, be some numbers and $1 \leq j_l \leq n^{(l)}$, $1 \leq l \leq d$, be some indices. The points

$$x_{j_1,\ldots,j_d} = (x_{j_1}, \ldots, x_{j_d})^T \in \Omega \cup \partial\Omega \tag{1.2}$$

*are called **grid points**. These are points where intensities of the image are sampled. A* **d-*dimensional grid array***,

$$X = \left(x_{j_1,\ldots,j_d}\right)_{j_l=1:n^{(l)},\ l=1:d} \in \mathbb{R}^{n^{(1)}\times\ldots\times n^{(d)}}, \tag{1.3}$$

*is the array of all the grid points of the image. It represents an $n^{(1)} \times \ldots \times n^{(d)}$ **grid**, that is the set $\Omega_d = \{x_j,\ j = 1 : N\}$ where $N = \prod_{l=1}^{d} n^{(l)}$. Numbers $1 \leq j \leq N$ are related to the index vectors $(j_1, \ldots, j_d) \in \mathbb{N}^d$, by the one-to-one lexicographical ordering given by*

$$j = \sum_{\nu=1}^{d-1}(j_{\nu+1} - 1)\prod_{\mu=1}^{\nu} n^{(\mu)} + j_1. \tag{1.4}$$

*The **grid vector** is the unfolding vector $\mathbf{X} = (x_j)_{j=1:N} \in \mathbb{R}^N$.*

Relating this to interpolation techniques, one needs to know how the intensities are positioned in the spatial domain. Two types of grids are presented here: *the cell-centered grid* $\mathbf{X}_c$ (c for centered) [4] and *the nodal grid* $\mathbf{X}_n$ (n for nodal).

3. A compact set where $I(x)$ is not zero is called support of I.
4. In this work, the term *cell* is used for both segments, cells and boxes.

Definition 1.3 (*Cell-centered and nodal grids*)

Consider $\Omega = \times_{l=1}^{d}[w_{2l-1}, w_{2l}]$. *A* ***cell-centered grid*** *is a partitioning of the space into congruent cells or boxes. Mathematically, it is expressed by*

$$\mathbf{X}_c = \left\{ x_j = (\xi_{j_1}^{(1)}, \ldots, \xi_{j_d}^{(d)}) \in \mathbb{R}^d, j_l = 1 : n^{(l)},\ l = 1 : d \right\}, \tag{1.5}$$

where

$$\xi_j^{(l)} = \omega_{2l-1} + (j - 0.5)h^{(l)}, \quad \text{and} \quad h^{(l)} = (\omega_{2l} - \omega_{2l-1})/n^{(l)}. \tag{1.6}$$

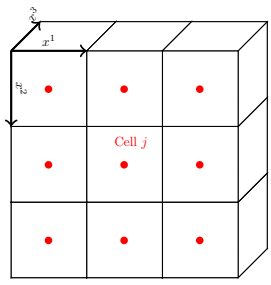
The d-dimensional intervals

$$cell_j = \{x_j = (x_j^{(1)}, \ldots, x_j^{(d)}) \in \mathbb{R}^d | -\frac{h^{(l)}}{2} < x_j^{(l)} - \xi_j^{(l)} < \frac{h^{(l)}}{2}\}$$

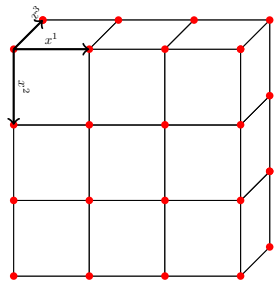
are called ***cells****, while the cell centers are* $x_j = (\xi_{j_1}^{(1)}, \ldots, \xi_{j_d}^{(d)})$.
A ***nodal grid*** *is defined by spatial nodes given by*

$$\mathbf{X}_n = \left\{ (h^{(1)} j_1, \ldots, h^{(d)} j_d) \in \mathbb{R}^d, j_l = 0 : n^{(l)},\ l = 1 : d \right\}. \tag{1.7}$$

Cell-centered and nodal grids are illustrated in Figure 1.4. The cell-centered grid illustrated in Figure 1.4a, considers that intensities are sampled at centers of given boxes or intervals (red points). One alternative of the cell-centered grid is the nodal grid 1.4b. In the nodal grid, intensities of the image are considered as a sample at the mesh points (red points).



(a) 3D cell-centered grid: intensities are sampled at the center of the boxes (red points)



(b) 3D nodal grid: intensities are sampled at the mesh points (red points)

Figure 1.4: Cell-centered and Nodal grid models

In most cases, one may be interested in connecting grid concepts and numbering to discretization of partial differential equations (PDEs). This has to

take into account the boundary conditions or initial conditions. For this purpose, three types of grids are defined below:

Definition 1.4

Let $d \in \mathbb{N}$, $\Omega = [0\ 1]^d$, and some positive integers $n^{(1)}, \ldots, n^{(d)}$. For $1 \leq j_l \leq n^{(l)}$, $1 \leq l \leq d$ related to j by the relation (1.4), the grid points

$$x_j^D = \left(\frac{j_1}{n^{(1)}+1}, \ldots, \frac{j_d}{n^{(d)}+1} \right)^T, \tag{1.8}$$

$$x_j^N = \left(\frac{2j_1-1}{2n^{(1)}}, \ldots, \frac{2j_d-1}{2n^{(d)}} \right)^T, \tag{1.9}$$

$$x_j^P = \left(\frac{j_1-1}{n^{(1)}}, \ldots, \frac{j_d-1}{n^{(d)}} \right)^T, \tag{1.10}$$

*constitute the grids Ω_D, Ω_N and Ω_P called respectively **Dirichlet, Neumann, and Periodic grids**.*

One can observe that, for $\Omega = [0\ 1]^d$ cell-centered grids are *Neumann grids*. In this work, the focus is on these Neumann grids.

1.3.2 Cubic spline interpolation

The objective is to find a continuous function $I(x)$ called *interpolant*, defined everywhere in the image domain Ω using known samples in this domain (cell-centered points). This function interpolates the measurements $dataI \in \mathbb{R}^N$ at some cell-centers $x_j, j = 1 : N$ and it is assumed that the function vanishes outside the image support domain. In variational settings, we minimize the distance

$$E_{D,I}(I(x), dataI) = \frac{1}{2} \| I(x) - dataI \|^2_{\mathbb{R}^N}, \tag{1.11}$$

under the constraints

$$I(x_j) = dataI(j), \quad j = 1 : N \ \text{ and } \ I(x) = 0, \ \forall\, x \notin \Omega. \tag{1.12}$$

The condition (1.12) is called *interpolation conditions*. In this work, the considered function is a cubic spline. The unidimensional case is explained here, while the multidimensional case is a direct generalization of the unidimensional case, grace to Kronecker product [10].

The function from cubic spline interpolation is a linear combination of cubic spline bases functions (B-splines). That is

$$I(x) = \sum_{j=1}^{N} c_j b^j(x), \tag{1.13}$$

where $b^0(x)$ is called "mother" spline and is given by:

$$b^0(x) = \begin{cases} (x+2)^3 & -2 \le x < -1 \\ -x^3 - 2(x+1)^3 + 6(x+1) & -1 \le x < 0 \\ x^3 + 2(x-1)^3 - 6(x-1) & 0 \le x < 1 \\ (2-x)^3 & 1 \le x < 2 \\ 0 & \text{everywhere else,} \end{cases} \tag{1.14}$$

and the other B-splines are defined by translation of $b^0(x)$ (also denoted simply $b(x)$). That is, for $j \in \mathbb{N}$,

$$b^j(x) = b^0(x-j), \quad \forall\, x \in \mathbb{R}. \tag{1.15}$$

We now need to know the intensity at any point in the domain. From (1.6), we have $\xi_j^{(1)} = \omega_1 + (j-0.5)h^{(1)}$ for $l = 1$ and we may omit $l = 1$ for this unidimensional case. In addition, setting $\Omega = [0\ 1])$, the cell centers are defined by $\xi_j = h(j-0.5)$. Let us apply a linear transformation $\xi \to x = \xi/h + 0.5$. With this transformation, cell centers become $x_j = j$, $j = 1, \ldots, N(= n^{(1)})$ while non cell centers $x \in \mathbb{R}$ can be writen in the form

$$x = j + \eta, \quad 0 < \eta < 1, \tag{1.16}$$

where j is the greatest integer that is less than x. As $b(x) = 0\ \forall\, x \notin [-2, 2[$ for the jth cell, $b^j(x) \neq 0$ only if $x \in [j-2, j+2[$. That is, at least four basis functions will not vanish in each cell. For each cell j, the function (1.13) becomes:

$$I(x) = c_{j+2}b^{j+2}(\eta - 2) + c_{j+1}b^{j+1}(\eta - 1) + c_j b^j(\eta) + c_{j-1}b^{j-1}(\eta + 1). \tag{1.17}$$

To determine the coefficients $\mathbf{c} = (c_1, \ldots, c_N)$, one evaluates (1.17) at the N cell-centers while imposing the interpolation constraints (1.12), see [10]. Since one can write any point as a sum of an integer and a reminder (see 1.16), the evaluation of (1.17) gives rise to a matrix B of size $N \times N$:

$$B = (b_j^k)_{j=1:N,k=1:N} = \begin{bmatrix} 4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{bmatrix} \in \mathbb{R}^{N \times N}. \tag{1.18}$$

The linear system

$$B\mathbf{c} = dataI \tag{1.19}$$

remains to be solved. Its solution gives the coefficients $\mathbf{c}$.

In the multidimensional case, we assume the separability [5] condition. Then, the interpolation function is a Kronecker product of unidimensional functions

$$I(x) = \sum_{j_d=1}^{n^{(d)}} \cdots \sum_{j_1=1}^{n^{(1)}} c_{j_1,\ldots,j_d} b^{j_1}(x^{(1)}) \cdots b^{j_d}(x^{(d)}), \tag{1.20}$$

5. Here, a multivariate function is considered separable if its Hessian is diagonal. This allows to approximate the multivariate function by Kronecker product of univariate functions.

where $x = (x^{(1)}, \ldots, x^{(d)})$.

Let us recall that the Kronecker product of two matrices $A \in \mathbb{R}^{n^{(1)} \times n^{(2)}}$ and $B \in \mathbb{R}^{m^{(1)} \times m^{(2)}}$ is computed by :

$$A \otimes B = \begin{bmatrix} a_{11}B & \ldots & a_{1n_2}B \\ \vdots & \ddots & \vdots \\ a_{n_1 1}B & \ldots & a_{n_1 n_2}B \end{bmatrix} \in \mathbb{R}^{n^{(1)}m^{(1)} \times n^{(2)}m^{(2)}}, \tag{1.21}$$

In this case, $I(x)$ in (1.20) can be written :

$$I(x) = I^{(d)}(x) \otimes \ldots \otimes I^{(1)}(x), \tag{1.22}$$

and it can be shown (see [10]) that the matrix of the linear system to be solved writes $\mathbf{B} = B^{(d)} \otimes \ldots \otimes B^{(1)}$.

This cubic spline interpolation is known to be the best compromise between efficacity (in term of computation cost) and differentiability (required for using variational methods for optimization). However, one can observe oscillation behaviour even where data may be constant. This means that, the obtained continuous image by this interpolation is not sufficiently regular and needs to be regularized.

Convolution methods may be used to regularize this function, using predefined kernels. But the drawback of such methods is that they do not take into account local information within the image. All points are equally treated. The preferred methods use local information such as the gradient or the curvature norm at each point. These are the regularization methods used in this work.

1.3.3 Regularization of the interpolant

The regularization of the interpolant aims to deal with the question of whether we need a function that fits the data perfectly but may be less regular or a more regular function that matches the data as well as possible. We may not need a perfect fit of the data since these data are contaminated by noise and are samples of more regular phenomena.

One way of regularizing the interpolant is to constrain the bending energy of the interpolant

$$E_{R,I}(I(x)) = \frac{1}{2} \int_\Omega (I"(x))^2 dx, \tag{1.23}$$

to be minimized [10]. Here $I"(x)$ expresses the second derivatives of the interpolant $I(x)$. The interpolation problem (1.11) becomes :

$$\min_I E_{R,I}(I(x)) \quad \text{subject to} \quad E_{D,I}(I(x), dataI) = 0. \tag{1.24}$$

In this case, the fit of the data is a constraint and the regularity is relaxed. On the other hand, one can insist on regularity and relax the fit. That is to solve the problem

$$\min_I E_{D,I}(I(x), dataI) \quad \text{subject to} \quad E_{R,I}(I(x)) = 0. \tag{1.25}$$

The regularization used in this work is a compromise obtained when (1.24) and (1.25) are combined. That is to solve the problem

$$\min_I \left(E_{D,I}(I(x), dataI) + \lambda E_{R,I}(I(x)) \right), \tag{1.26}$$

where $\lambda \geq 0$ is a weighting parameter, called *Tychonoff parameter*, that allows a balance between data fitting and regularity.

Since the function $I(x)$ is parametrizable, the computations can be performed in the parameter space to get coefficients of the regularized interpolant. Thus, instead of solving (1.19), one minimizes (1.26) with,

$$E_{D,I}(I(x), dataI) = \frac{1}{2}\|B\mathbf{c} - dataI\|^2_{\mathbb{R}^N} \tag{1.27}$$

and

$$E_{R,I}(I(x)) = \frac{1}{2}\|\mathbf{c}\|^2_M = \frac{1}{2}\mathbf{c}^T M\mathbf{c}. \tag{1.28}$$

The matrix M is computed evaluating

$$M_{j,k} = \int_\Omega b"^j(x) b"^k(x) dx \quad 1 \leq j,k \leq N. \tag{1.29}$$

This can be seen by minimizing (1.23) in the L^2-norm. Coefficients c are thus solution of the minimization problem

$$\min_{\mathbf{c}} \left(\frac{1}{2}\|B\mathbf{c} - dataI\|^2 + \frac{\lambda}{2}\mathbf{c}^T M\mathbf{c} \right), \tag{1.30}$$

whose minimizer is the unique solution of

$$(B^T B + \lambda M)\mathbf{c} = B^T dataI. \tag{1.31}$$

In a more general approach, the matrix M can be replaced by an arbitrary symmetric positive semidefinite weighting matrix P. This gives

$$(B^T B + \lambda P)\mathbf{c} = B^T dataI, \tag{1.32}$$

that also allows a unique solution. When P is chosen to be the identity matrix I_N, the regularization is the so-called *Tychonoff regularization.* The *Tychonoff-Phillips regularization* uses $P = D^T D$ where

$$D = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{N+1 \times N} \tag{1.33}$$

approximates the first derivatives. $P = M$ is theoretically justified since the optimization is in the spline space (function that minimizes the bending energy).

In multidimensional settings, the generalization is direct using Kronecker products: $\mathbf{B} = \bigotimes_{j=1}^{d} B^j$, $\mathbf{P} = \bigotimes_{j=1}^{d} P^j$, $\mathbf{M} = \bigotimes_{j=1}^{d} M^j$ and $\mathbf{C} = \bigotimes_{j=1}^{d} \mathbf{c}^j$, and solving

$$(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{P})\mathbf{c} = \mathbf{B}^T data I. \tag{1.34}$$

1.3.4 Derivatives of the interpolant

The interpolant $I(x)$ given by (1.20) can be written using the separability assumption:

$$I(x) = \sum_{j_d=1}^{n^{(d)}} \cdots \sum_{j_1=1}^{n^{(1)}} c_{j_1,\ldots,j_d} b^{j_1}(x^{(1)}) \cdots b^{j_q}(x^{(q)}) \cdots b^{j_d}(x^{(d)}). \tag{1.35}$$

Then, its first order partial derivatives are obtained via derivatives of B-splines by the formulae

$$\partial_q I(x) = \sum_{j_d=1}^{n^{(d)}} \cdots \sum_{j_1=1}^{n^{(1)}} c_{j_1,\ldots,j_d} b^{j_1}(x^{(1)}) \cdots (b^{j_q}(x^{q}))' \cdots b^{j_d}(x^{(d)}), \tag{1.36}$$

where $(b^{j_q}(x^{(q)}))'$ may be computed from (1.14). From discrete point of view, let us consider the cell-centered grid with points $x_j \in \mathbb{R}^d$, $j = 1 : N$, that is $x_j = (x_j^{(1)}, \ldots, x_j^{(d)})$. But all these coordinates can be stored in a long vector $X \in \mathbb{R}^{Nd}$ of the form :

$$X = [x_1^{(1)}, \ldots, x_N^{(1)}, x_1^{(2)}, \ldots, x_N^{(2)} \ldots; x_1^{(d)}, \ldots, x_N^{(d)}]^T. \tag{1.37}$$

In this formulation, the interpolant can be seen as the function I: $\mathbb{R}^{Nd} \to \mathbb{R}^N$. It is defined such that its j^{th} component is given by $I_j(x) = I(x_j) = I(x_j^{(1)}, ..., x_j^{(d)})$ and the Jacobian matrix obtained by the relation (1.36) is :

$$dI(x) = \left[\frac{\partial I_j(x)}{\partial x^{(k)}}\right]_{j=1:N;\ k=1:Nd} \in \mathbb{R}^{N\times Nd}, \tag{1.38}$$

obtained by the relation (1.36). Let us note $\dfrac{\partial I_j^{(k)}(x)}{\partial(x_j^{(k)})}$ the derivatives of the k^{th} component with respect to the j^{th} variable in coordinate k, $j = 1 : N$, and $k = 1 : d$. Since the j^{th} component $I_j(x)$ depends only on $x_j = (x_j^{(1)}, ..., x_j^{(d)})$, the Jacobian is a block matrix with diagonal blocks of the form:

$$dI(x) = \begin{bmatrix} \frac{\partial I_1^{(1)}(x)}{\partial (x_1^{(1)})} & & \frac{\partial I_1^{(2)}(x)}{\partial (x_1^{(2)})} & \cdots & \frac{\partial I_1^{(d)}(x)}{\partial (x_1^{(d)})} \\ & \ddots & & \ddots & & \ddots \\ & & \frac{\partial I_N^{(1)}(x)}{\partial (x_N^{(1)})} & & \frac{\partial I_N^{(2)}(x)}{\partial (x_N^{(2)})} & \cdots & \frac{\partial I_N^{(d)}(x)}{\partial (x_N^{(d)})} \end{bmatrix}.$$

1.3.5 Variational formulation of the MIRP

Definition 1.5

The image registration problem aims to find a suitable spatial transformation *that deforms one or more images such that, each transformed image becomes* as similar as possible *to one fixed image, considered as the reference image.*

Formally, consider I_f and I_m, two d-dimensional continuous images (obtained after interpolation). Let I_f be fixed, while I_m is under deformations. The registration problem aims to solve the minimization problem :

$$\hat{\Phi} = argmin_{\Phi \in \mathcal{T}} E_D(I_f, I_m, \Phi), \tag{1.39}$$

where E_D is a criteria that quantifies the dissimilarity between the two images I_f and I_m while $\mathcal{T}$, the search space, is a set of admissible transformations $\Phi : \mathbb{R}^d \to \mathbb{R}^d$. Then the functional to be minimized writes

$$\mathcal{F}(\Phi) = E_D(I_f, I_m, \Phi). \tag{1.40}$$

In this thesis, the "*discretize and then optimize*" approach is used. It consists of replacing the continuous problem by a discrete problem taking into account the required smoothness (regularization). Then, appropriate optimization methods lead to a solution sufficiently close to that of the continuous problem. The Medical Image Registration Problem (MIRP) is an iterative process, as shown in Figure 1.5, where the major work is done in the optimization loop [26]. In this diagram, we propose that the fixed image be interpolated once. This is not important and may be skipped since we neither use derivatives of the fixed image nor its continuous form. However, as we use regularized interpolation, this can help to regularize and thus to reduce noise in the image. Conversely, deforming the moving image is equivalent to re-interpolating it in the deformed grid. This is the reason why it is interpolated after each new transformation from the optimization loop. Iterations are stopped when the dissimilarity measure becomes small with respect to some predefined threshold.

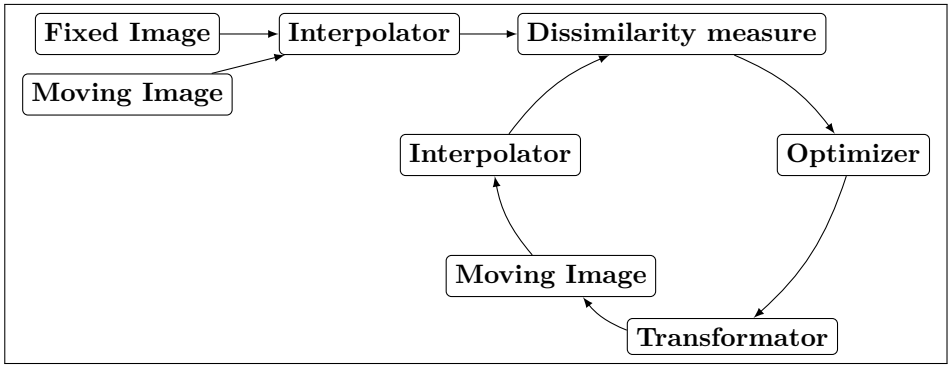


Figure 1.5: The MIRP considered as an itevative process.

The problem (1.39) is ill-posed according to Hadamard [6], because the uniqueness of its solution is not guaranteed. Indeed, one wants to determine a vector $\Phi(x) \in \Omega \subset \mathbb{R}^d$ from scalar information $I_m(x),\ \forall x \in \Omega \subset \mathbb{R}^d$. Hence, one solves an underdetermined problem. Therefore, there is a need to add another term that regularizes the problem, adding information about the sought transformation (see for example [10], page 118, [27] and [28]).

1.3.6 Regularizing the problem

The regularization of the interpolant is addressed in subsection 1.3.3. Here, the same idea continues but the condition of regularity is rather imposed on a transformation that is not parametric. The regularization term E_R is added to the problem (1.39), such that it becomes well-posed. This leads to the problem:

$$\min_{\Phi \in \mathcal{T}} E_D(I_f, I_m, \Phi) + \lambda E_R(\Phi), \tag{1.41}$$

where $\lambda \in \mathbb{R}_+$ is the Tychonoff parameter, that defines the weight of the regularity with respect to the dissimilarity measure. Thus, the cost function (1.40) is formulated as

$$\mathcal{F}[\Phi] = E_D(I_f, I_m, \Phi) + \lambda E_R(\Phi). \tag{1.42}$$

The term $E_R(\Phi)$ imposes the function Φ to be sufficiently regular, that is, it should be bijective (and consequently invertible) and continuously differentiable. In other words, we will choose in the space $\mathcal{T}$ only the transformations that better satisfy the criterion of regularity defined by the regularization term E_R. In this work we focus on the transformations which minimize a certain energy constructed from local information, within the gradient (the Jacobian matrix) or the Laplacian (See [28]).

6. See the mathematics encyclopedia `https://www.encyclopediaofmath.org/index.php/Ill-posed_problems`.

Different choices related to the dissimilarity criteria E_D, to the set of transformations $\mathcal{T}$ and the optimization strategy are allowed. These choices are generally guided by the application at hand, as well as by the nature and the methods of the images acquisition. This leads to several classes of registration methods (see [1],[2]). In the following subsection, main dissimilarity measures are briefly presented.

1.3.7 Dissimilarity measures in MIRP

Let us consider I_f and I_m respectively the fixed and the moving images. Many ways exist to measure dissimilarities within images. However, any dissimilarity measure may depend on some assumptions about the relationship between the concerned images [29]. The most used dissimilarities or similarities in medical image registration are listed below (see [10] for more details).

The Sum of Squared Differences (SSD)

Let us assume that gray values of corresponding points in the grid also correspond, that is

$$I_m[x] = I_f(x),\ \forall\, x \in \Omega.$$

This assumption is called the constancy assumption. With this the assumption we can measure the dissimilarity by

$$E_D^{SSD} = \int_\Omega (I_m[\Phi(x)] - I_f(x))^2 dx. \tag{1.43}$$

Another measure of a such consideration is the *Sum of Absolute value Differences* (SAD) of intensities $E_D^{SAD} = \int_\Omega |I_m[\Phi(x)] - I_f(x)|dx$. However, note that this assumption is most reliable and applicable if images are captured under the same conditions and, in general, with the same modalities.

Normalized Gradient Fields (NGF)

In practice, the constancy assumption after which gray values of corresponding points are equal appears rather restrictive and not applicable for multimodal images [29]. An extension should take into account that contents of the moving image I_m is also displayed by intensity changes expressed by the image gradient ∇I_m. Therefore, the image gradient has to play a dominant role in dissimilarity measure. This leads to dissimilarity measures based on gradients, with general assumption

$$\nabla I_m[\Phi(x)] = \nabla I_f(x),\ \forall\, x \in \Omega. \tag{1.44}$$

Since the gradient is orthogonal to the level set $L(c) = \{x : I_m(x) = c\}$, this assumption can be interpreted as capturing the misalignment of level sets [10]. However, the gradient also measures the strength of change and this is not

a desirable information for multimodal registration. Therefore, the assumption (1.44) has to be replaced by

$$\frac{\nabla I_m[\Phi(x)]}{\|\nabla I_m[\Phi(x)]\|} = \frac{\nabla I_f(x)}{\|\nabla I_f(x)\|}, \tag{1.45}$$

assuming $\nabla I_m[\Phi(x)], \nabla I_f(x) \neq 0$. Moreover, the image noise may expand by the normalization step, to corrupt the reliable information given by the gradient field. A parameter η is needed, called *edge parameter* that determines a reliable edge $|\nabla I_m| > \eta$ from those contaminated by noise $|\nabla I_m| < \eta$. Noting $I_m[\Phi(x)] = I_m$, the normalized gradient of I_m is then given by

$$n[I_m] = n[I_m, \eta] = \frac{\nabla I_m}{\sqrt{|\nabla I_m|^2 + \eta^2}}$$

and the similarity measure writes

$$E_D^{NGF} = \int_\Omega (n[I_m]^T n[I_f(x)])^2 dx. \tag{1.46}$$

If no bias is to be feared, the ideal situation is such that the two gradient fields are aligned. This means one maximizes (1.46). However, maximizing (1.46) is equivalent to minimizing

$$E_D^{NGF} = \int_\Omega 1 - (n[I_m]^T n[I_f])^2 dx. \tag{1.47}$$

Therefore, one minimizes the misalignment of the two images by minimizing (1.47).

Normalized cross-correlation (NCC)

Another way of passing over the constancy assumption is to assume a linear relationship between intensities of I_m and I_f [29]. This point of view assumes that a linear function f exists such that

$$I_m[\Phi(x)] = f(I_f(x)),\ x \in \Omega. \tag{1.48}$$

In this context, dissimilarities are measured by the cross-correlation between the fixed image and a linearly transformed version of the moving image [10]. From the fact that

$$(I_m[\Phi(x)] - I_f(x))^2 = I_m[\Phi(x)]^2 - 2I_m[\Phi(x)]I_f(x) + I_f(x)^2,$$

one can observe that the right hand side of the equation above is minimized when its second term $2I_m[\Phi(x)]I_f(x)$ is maximized. Therefore, we are interested in the cross-correlation given by

$$\langle I_m[\Phi(x)], I_f(x)\rangle = \int_{\mathbb{R}^d} I_m[\Phi(x)]I_f(x)dx.$$

However, the transformation may enable scaling of the moving image and lead to biased measures. The commonly used similarity measure to be maximized is, therefore, the normalized cross-correlation:

$$NCC[I_m, I_f] = E_D^{NCC}[I_m, I_f] = \frac{\langle I_m[\Phi(x)], I_f(x)\rangle}{\|I_m[\Phi(x)]\|\|I_f(x)\|}, \tag{1.49}$$

where $\|I_m\| = \sqrt{\langle I_m, I_m\rangle}$ and $I_f(x), I_m[\Phi(x)] \neq 0$. In the minimization formulation, one has to minimize $E_D^{NCC} = 1 - NCC[I_m, I_f]^2$.

Normalized Mutual Information (NMI)

Mutual information is the most commonly used similarity measure in multimodal image registration. With the observation that : "*A combined version of two misaligned images should be more complex than the combined version of the same two images when they are aligned*"([29] p. 16), the main idea is then to maximize the normalized entropy of the joint density of $I_m[\Phi(x)]$ and $I_f(x)$ [10]. Let us define a histogram summarizing all possible correspondences in a sequence of N pairs of intensities by

$$\rho^{hist}(t, r) = \#\{(I_f^i, I_m^j) | I_f^i = t \text{ and } I_m^j = r\}/n, \;\; i, j = 1 : N. \tag{1.50}$$

This is the number of all voxels whose intensities are equal to t in I_f and equal to r in I_m. Given a histogram ρ^{hist}, its joint entropy H may be written as:

$$H[\rho^{hist}] = -\sum_{t,r} \rho^{hist}(t, r) log \rho^{hist}(t, r) = -\frac{1}{n} \sum_{i,j} log \rho^{hist}(I_m^i, I_f^j). \tag{1.51}$$

Defining the marginal densities

$$\rho_{I_f}^{hist}(t) = \sum_r \rho^{hist}(t, r), \quad \text{and} \quad \rho_{I_m}^{hist}(r) = \sum_t \rho^{hist}(t, r),$$

the normalized mutual information to maximize is measured by

$$E_D^{NMI}[\rho^{hist}] = H[\rho_{I_f}^{hist}] + H[\rho_{I_m}^{hist}] - H[\rho^{hist}]. \tag{1.52}$$

See [10, 29] for more details on mutual information and normalized mutual information measures.

Once the dissimilarity measure is chosen, we need to fix also the search space $\mathcal{T}$. This informs on the nature of the sought transformation. The most used transformations in image registration are shortly recalled below.

1.3.8 Transformations in MIRP

The type of spatial transformation or mapping used to perfectly overly two images is one of the fundamental characteristics of any image registration technique [2]. In this section, the most used transformations are presented briefly

while the focus is on deformable or non-rigid and non-parametric transformations.

Definition 1.6 (*Degree of freedom*)

The degree of freedom (DOF) of a transformation is the number of independent parameters needed to represent this transformation.

The degree of freedom also characterizes the transformation type. The higher is the degree of freedom, the more the transformation can express complex changes but more regularization is also required. Table 1.2, that is a rough modification of a table from [30, Table 2.1], presents the commonly used transformations and their degree of freedom in 3D. These transformations may be

Type	**DOF(m)**	**Remarks**	Parametric
Rigid	6	Applicable to rigid structures	yes
Affine	12	Coarse approximation of non-rigid transformation	yes
Polynomial	$3\sum_{i=1}^{p} C_{i+2}^{0}$	Non-rigid approximation, p is the polynomial degree	yes
Piece-wise affine	$12N$	Trade-off DOF/non-rigidity, N the number of affine pieces	yes
Free-Form Deformation	$3N_xN_yN_z$	Non-rigid, (N_x, N_y, N_z) is the grid size	no
Radial basis functions	$3N$	N is the number of nodes, non-rigid	yes
Deformation maps	3N	N is the number of deformation vectors	no

Table 1.2: Main transformations and their Degree of Freedom (DOF).

classified into parametric and non-parametric transformations.

Parametric transformations

Let us denote by $m \in \mathbb{N}$ the degree of freedom of a given transformation. A transformation is parametric, when it can be expanded as combination of some parameters a_j and basis functions φ_j $(j = 1 : m)$. Thus, the required transformation is a minimizer of a defined distance measure in the space spanned by these basis functions [2]. The most used parametric transformations are rigid transformations, affine transformations and polynomials transformations in a general sens (this iclude piece-wise polynomials).

A rigid transformation reffers to a combination of rotation, translation and a scale change (global zoom).

An affine transformation reffers to a more global than rigid transformation since it admits rotation, translation, shearing [7] and individual scaling (local zoom) [10, p.49].

A polynomial transformation expresses higher variability by increasing its degree. This means that increasing the degree does not always lead to more accurate results. In addition, the more a transformation is rigid, less it has degree of freedom but less it is accurate since it can not treat local information. Conversely, the more a transformation is non-rigid, the more it is accurate but the more it is hard to compute. *Piecewise polynomial transformations* are proposed to give a compromise between the degree of freedom and the non-rigidity (possibility to transform local elements). They include piecewise affine transformations, spline-based transformations, and more "unusual" polynomials [10].

Wavelets transformations are parametric transformations that use linear combinations of wavelets basis functions. They mostly used in image reconstruction.

The drawback of parametric transformations is that the basis is chosen artificially. Then, it remains difficult to justify one choice rather than another. To overcome this drawback, non-parametric transformations have been addressed. Below is a presentation of the general framework of non-parametric transformations.

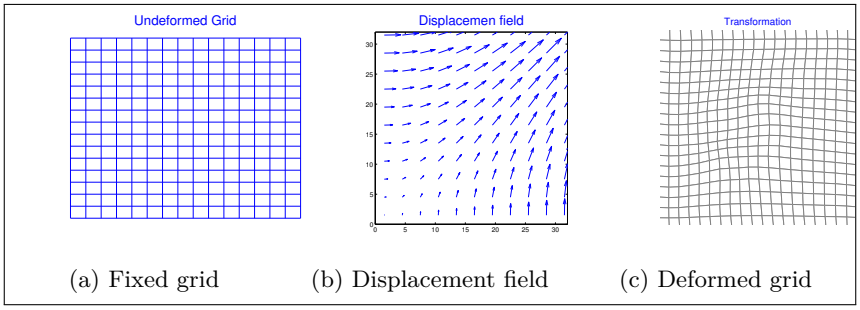
Non-parametric transformations

Non-parametric transformations are naturally non-rigid. In non-parametric settings, the transformation is applied to each voxel in the image domain. Thus, the degree of freedom depends on the number of voxels and the dimension of the space. In 3D, as each voxel is determined by 3 coordinates, the degree of freedom is three times the number of voxels in the image. Commonly, at each point $x \in \Omega$, the non-parametric transformation is given as addition of the identity with a displacement field. That is,

$$\Phi(x) = (Id + u)(x) = x + u(x), \text{ where } u : \mathbb{R}^d \to \mathbb{R}^d, \tag{1.53}$$

is the displacement field also called the *deformation*. Observe that, the transformation is explicitly determined by the displacement field u. Thus, in the rest, u and Φ may be interchanged if no confusion is possible. Figure 1.6 illustrates the displacement field in 2D. Given a fixed grid, one computes some small numbers with their orientations (sign) to get a deformed grid by addition.

7. The shear transformation, called shearing, displaces each point in a fixed direction by an amount proportional to its signed distance from a line that is parallel to that direction `https://en.wikipedia.org/wiki/Shear_mapping`.



(a) Fixed grid (b) Displacement field (c) Deformed grid

Figure 1.6: Illustration of a deformation in a non-parametric transformation.

Theoretically, The transformation $\Phi(x)$ is assumed to map homologous locations in the moving image physiology to the reference image physiology (forward mapping, see Figure 1.7a). However, in practice and from an implementation point of view, the sought transformation has to map homologous locations from moving image physiology in the reference image physiology and interpolate therein (backward mapping). Mathematically, instead of evaluating $I_m[\Phi(x)] - I_f(x)$ one evaluates $I_m(x) - I_f[\Phi^{-1}(x)]$ that are equivalent for invertible Φ.

Figure 1.7b illustrates the reasons why it is not advisable to use the forward transformation. One computes a new physical position for the moving image and needs to interpolate intensities from fixed image therein (forward transformation). However, since the grid of the moving image is under deformations, the spacing may not still be regular and interpolation of such intensities may cause numerical instabilities. To overcome this, the backward approach computes new positions while the interpolation is still done in the grid of the fixed image that remains regular. In brief, Forward transformations interpolate in a deformed grid (irregular spacing), while backward transformations interpolate in the initial fixed grid where spacing remains regular.

To compute this inverse, one can assume $\|u\|$ sufficiently small such that

$$(id + u)^{-1} = id - u, \tag{1.54}$$

and

$$\Phi(x) = x + u(x) \quad \text{and} \quad \Phi^{-1}(x) = x - u(x). \tag{1.55}$$

Regularization techniques of non-parametric transformations have been designed and used to dictate the nature of the deformation model [8] u. In general, three main classes of transformation models are distinguished: those inspired by physical models, those inspired by interpolation and approximation theory,

8. Variational approaches in general attempt to determine a function, not just a set of parameters ([1]).

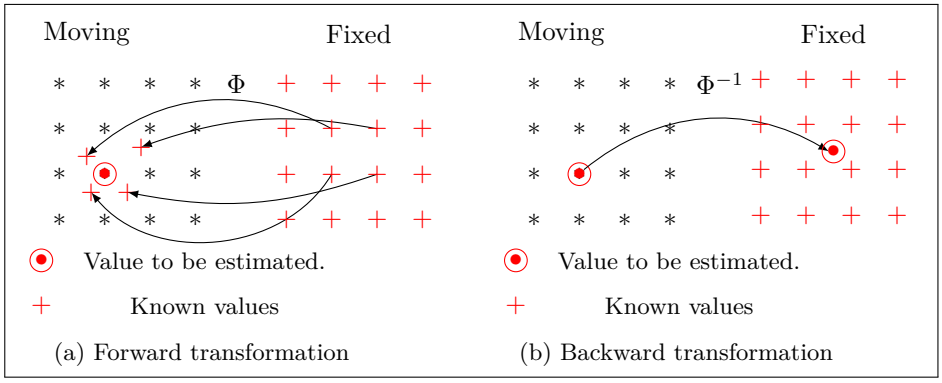


Figure 1.7: Forward Vs backward transformations.

and those called knowledge-based deformation models (that use prior information regarding the sought deformation). To these are added models that satisfy a specific task constraint [1].

Without being exhaustive, a short overview on different spatial transformations derived from physical models is presented. In this context, the deformable image registration is modelled as a physical process described by one or more partial differential equations (PDE). The resolution of these PDE by numerical methods (Finite Element Methods, Finite Differences Methods , Finite Volumes Methods or characteristics methods) gives the desired transformation. The following short descriptions are based on references [24], [1] and [2].

Elastic Body Models

Proposed by C. Broit in [31], elastic model considers the image under deformation as an elastic body. Let

$$u : \mathbb{R}^d \to \mathbb{R}^d, \text{ with } u^{(j)} \in C^2(\mathbb{R}^d), \ \forall \ j = 1 : d, \tag{1.56}$$

be a displacement field where $C^2(\mathbb{R}^d)$ is the set of twice continuously differentiable functions on $\mathbb{R}^d$. Then it is imposed to the sought transformation to be the one that minimizes the potential energy of linear elasticity given by:

$$E_R(u) = \frac{1}{2}\int_\Omega \mu_e(trace(\nabla u^T \nabla u) + \lambda_e(trace(\nabla u)^2)dx, \tag{1.57}$$

where the parameter μ_e refers to the rigidity and quantifies the stiffness of the material while λ_e is the Lamé first coefficient (no physical interpretation but serves for simplification in some expression describing Hooke's law, see [24] and [32]).

The first-order optimality condition $\nabla J[u] = 0$ of the function (1.42) when the dissimilarity measure is given by (1.43) and the regularisation term is given

by (1.57) (see [24]) leads to the so called Navier-Cauchy equation:

$$\mu_e \Delta(u) + (\lambda_e + \mu_e)\nabla(\nabla.u) + F = 0, \tag{1.58}$$

where the registration is led by the external force fields

$$F = (I_m - I_f)\nabla I_m, \quad I_m = I_m[x + u(x)],\ I_f = I_f(x), \tag{1.59}$$

which contain the residuals and derivatives of the deformed image. The equation that describes the deformation is (1.58). While external forces (1.59) are deforming the image, internal forces imposed by the elastic constraints of the body impose regularity of the displacement field until equilibrium.

Viscous Fluid Flow Models

In this type of model, the image under deformation is considered as a viscous fluid whose deformation is governed by the Navier-Stokes equation. The design is similar to the elastic model. Let F be given by (1.59), the Navier-Stokes equation writes:

$$\mu_f \nabla(v) + (\lambda_f + \mu_f)\nabla(\nabla.v) + F = 0. \tag{1.60}$$

Observe that this equation is similar to (1.58) where the displacement field is replaced by a velocity field and parameters λ_f and μ_f are adapted to the fluid context where they characterize viscosity coefficients. This model does not assume small deformations ($\|u\| <<$), it is able to recover large deformations [33]. The velocity field is related to the displacement field by the equation

$$v(x,t) = \frac{d}{dt}u(x,t), \tag{1.61}$$

that has to be integrated to approximate the displacement field u. This model is well suited for large deformations, for example when images concern fluid materials but are less appropriate for soft tissues (see [24, p. 136]). An important drawback remains the computational inefficiency unless parallel programming is used [33].

Diffusion model

The diffusion model considers the registration as a diffusion process governed by the general equation

$$\Delta u + F = 0, \tag{1.62}$$

where F is given by (1.59) and under appropriate initial and boundary conditions. Observe that, this equation can be derived from (1.58) for $\lambda_e = -\mu_e$ to a multiplicative constant, where Δ is a Laplace-like operator. The version proposed by J-P Thirion [4], is the most popular but the most controversial also, due to lack of sound theoretical justification ([1] and [24]). Its solution

may not even verify the equation (1.62).

The diffusion model has been subsequently formalized by [24] and proved to be particularly effective and fast. Thus, it remains popular in both professional and research communities although there is no acceptable physical interpretation for the image registration, see [24].

Curvature model

First-order models (Elastic, Fluid or Diffusion) need an affine pre-registration ([24], page 173). Applications for which a pre-registration can not be guaranteed use the curvature model. This model is governed by the equation

$$\Delta^2 u + F = 0. \tag{1.63}$$

It is a second-order model and therefore it admits a smoother transformation than the others. In addition, it is faster. However, it is also limited to small deformations and may not preserve anatomical topology of certain images.

Flow of diffeomorphisms model

In this model, the registration process is considered as a transportation of particles in a specific space. Then deformations are determined from their instantaneous velocities using the Lagrange transport equation [33]. The term of regularization is given by

$$E_R(\mathbf{v}) = \int_0^1 ||v_t||_V^2 dt, \tag{1.64}$$

where $||.||_V$ is a given norm defined on the space V of smooth velocity vector fields such that $||u||_V = ||Du||_{L^2}$ and D is a differential operator. This method admits large deformations and is often called *LDDMM (Large Deformation Diffeomorphic Metric Mapping)* This approach will be developed later. To learn more about these methods, see for xample [34] to [35]. Although this method produces a diffeomorphic transformation, it is not symmetric by construction. Other techniques have been developed for a diffeomorphic symmetric algorithm (see [36]).

1.3.9 Common issues in deformable MIRP

There are some constraints in medical image analysis that may appear common to be imposed to the transformation such that it exhibits special properties. Those constraints include:

Topology preservation

This criterion ensures that, during deformation, the connected structures remain so, and that the neighborhood and adjacency properties are preserved [28]. Thus, the transformation should not introduce cracks in the deformed image. Topology preserving algorithms produce a mapping that is continuous, locally one-to-one and that has a continuous inverse. The Jacobian determinant contains information regarding the injectivity of the mapping and has to be greater than zero. The differentiability of the transformation is needed to compute the Jacobian determinant.

Inverse consistency

The asymmetric behaviour of registration algorithms is such that, when interchanging reference and moving images, the algorithm does not estimate the inverse transformation. The aim of inverse consistency constraint is to tackle this asymmetric behaviour, constraining the forward and the backward transformation to be inverse mappings of one another. This requires the sought transformation to be continuous and invertible with continuous inverse. An algorithm that verifies this property should therefore verify the topology conservation. In this way one can switch from one image to another and vice versa. For other techniques to get such transformations, see for example the survey [1] and references therein.

Symmetry

As with inverse consistency, the symmetry constraint aims to tackle asymmetry of registration algorithms. It uses either a symmetric objective function by construction or estimates two transformations by simultaneously mapping the two images in a common domain. The final mapping from one image to another is calculated by inverting one transformation function and composing it with the other:

$$f \circ I_m = g \circ I_f \quad \text{and} \quad \Phi = g^{-1} \circ f, \quad \Phi^{-1} = f^{-1} \circ g.$$

Diffeomorphism

Diffeomorphic tranformations also preserve topology. In addition, they are invertible, at least theoretically, and both the diffeomorphic transformations and their inverse are differentiable. A diffeomorphism maps a differentiable manifold to another. Observe that, if the transformation is diffeomophic, it preserves topology and it is symmetric. This means that this constraint includes the others, at least theoretically.

Time and storage memory consuming

Currently, the need to register high resolution and high dimensional images is increasing. In particular, deformable medical image registration for $3D/4D$ high resolution images causes considerable problems. For example, images of size $[10^3, 10^3, 10^3]$ within a cell-centered grid, lead to linear systems of size $3.10^9 \times 3.10^9$ that need to be solved iteratively. This results in algorithms that need significant storage memory and computing time.

In addition, one can observe the so called "curse of dimensionality". This means that when the dimension is augmented, the computational cost increases exponentially with respect to the dimension. According to [5], almost all deformable 3D medical images registration algorithms face the problem of computing time. This computing time may be a significant issue for some clinical applications.

Although in many cases, when the registration quality is improved the computing time increases [34, 37], it would be naive to think that the more expensive an algorithm is, the better is the quality of registration. A good algorithm may try to reduce the computing time with little or no deterioration of the registration quality. This is possible when there is some a priori knowledge available, when using some properties of numerical operators and compression techniques and when using an efficient implementation in the process.

These issues are the main challenges for deformable medical image registration problems. This thesis mainly addresses the issue of time and memory consuming while we impose some criteria to the transformation to be at least locally diffeomorphic.

1.3.10 The cost function

We now need to build the cost function whose optimization will be the main task in the rest of this thesis. For this purpose, we have chosen to minimize the SSD dissimilarity measure (1.43) while the deformations are modelled following the elastic model. It will be shown that the diffusion model can be considered as a particular case of the elastic model. In these settings, the regularizations used are essentially based on the L^2-norm of the differential operators applied to the displacement field u. The cost function derived from the equation (1.42)considering the measure (1.43) and the regularizer (1.57) writes

$$\mathcal{F}[u] = \frac{1}{2}\|I_m[\Phi(x)] - I_f(x)\|_{L^2}^2 + \frac{\lambda}{2}\int_\Omega \|Bu\|_{L^2}^2 dx, \tag{1.65}$$

where

$$B = \mu_e\Delta + (\lambda_e + \mu_e)\nabla(\nabla), \tag{1.66}$$

is the differential operator and

$$\Phi = Id + u. \tag{1.67}$$

To optimize the function (1.65), the main techniques are addressed in chapter 2. Several image registration algorithms have been provided (see [1]) but most of them are designed for specific applications. More general libraries include the Insight Segmentation and Registration Toolkit (ITK [9]). However ITK is an advanced cross-platform developed through high programming methdologies in C++ language and this makes it less attractive for educational purposes. However, one of the most popular algorithm implemented in ITK is the Demons algorithm from [4]. In addition, a more educational package is the Flexible Algorithms for Image Registration (FAIR) provided by Jan Modersitzki [10] and this thesis follows the main methodology provided by this package. Below we present the main principles of FAIR algorithms (for deformable nonparametric registration) and principles of the Demons algorithms that have inspired this work.

1.4 Some registration algorithms

1.4.1 Flexible Algorithms for Image Registration (FAIR)

The FAIR package from Modersitzki [10] is a library of algorithms for medical image registration implemented in Matlab, whose object is educational and aresearch oriented.

It presents the registration process in an integrated and flexible approach (from storage to displaying results). In this library, the variational formulation of the registration problem, the discretization of different differential operators, the techniques of interpolation and the optimization of the functional are explained. In addition, a relatively efficient way of implementing each of these topics is provided, including a matrix-free framework. Concerning optimization, the emphasis is on Gauss-Newton like methods with line search strategy.

Linear systems are solved by iterative methods. These methods mainly use the Preconditioned Conjugate Gradient method with Jacobi or Gauss-Seidel preconditioning techniques. Other optimization methods such as trust-region methods are outlined but not sufficiently developed. A multilevel and multiscale approach is sufficiently developed and encouraged. Since deformable transformation is expensive, one may reduce the cost by applying first a rigid or affine transformation to be closer the solution. This is called pre-registration.

Figure 1.8 visualizes the main steps of a FAIR-like deformable registration. The optimization loop is the most expensive step.

9. `https://itk.org`

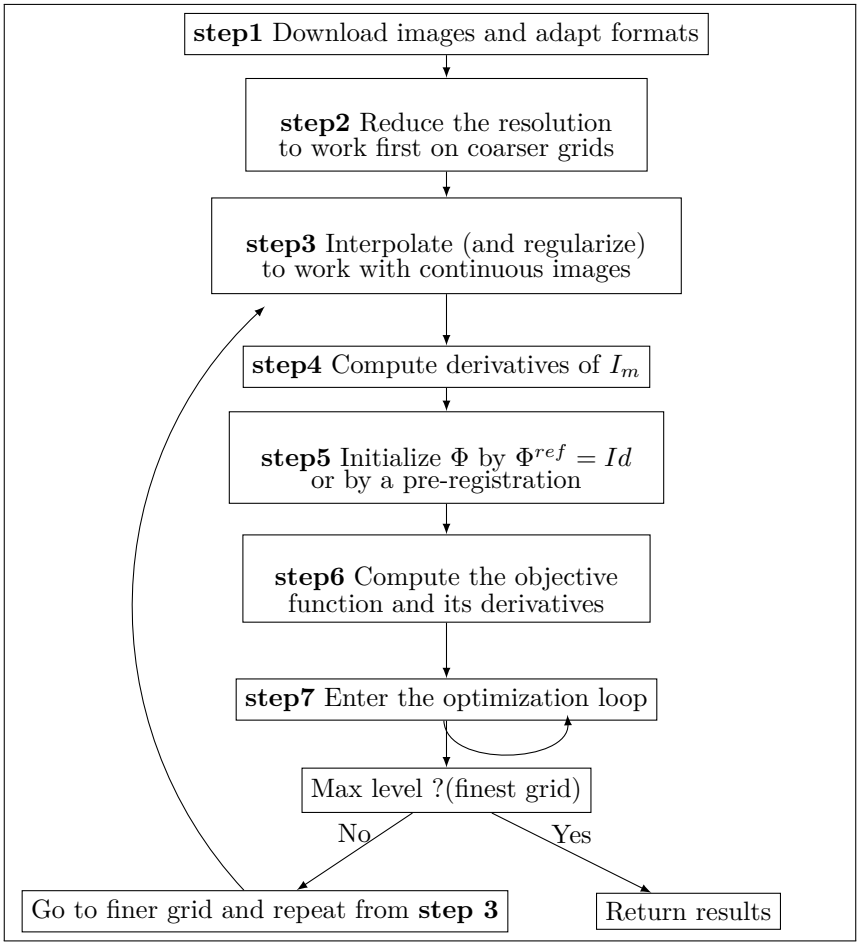


Figure 1.8: FAIR process: main steps in a multiresolution approach.

Multilevel approach

The multilevel approach is one of the most successful techniques in medical image registration, especially in high resolution or/and high dimensional images. Consider I_f and $I_m \in \mathbb{R}^{n^{(1)} \times n^{(2)} \times \ldots \times n^{(d)}}$, two images within two cell-centered grids in a common domain Ω. By fusing adjacent cells of these grids while averaging their intensities, more regular measurements are obtained and the size of the problem is halved. The resolution of the image is also halved.

Example 1.1

Example, for $k = 1 : d$, consider $n^k = 2^L$, $L \in \mathbb{N}$ and data $dataI_L^k = dataI^k \in \mathbb{R}^{n^k}$. In Matlab, *the representation of these data at level $l - 1$*

from data of level l (for $l = L : -1 : 1$) writes:

$$dataI_{l-1}^k = (dataI_l^k(1:2:n^k-1) + dataI_l^k(2:2:n^k))/2 \ \in \mathbb{R}^{n^k/2}.$$

Thus, one chooses to make the first registration for the data at the lowest level (for example $l = l_{min} = 3$ is the coarsest grid), and refines the results to the higher levels until reaching the initial resolution of the images at the finest grid.

FAIR objective function and derivatives

Again, the focus is on the SSD dissimilarity measure which is more commonly used in FAIR. For multimodal registration, we may argue a little on the NGF to link with the SSD.

Let us consider, once more, I_f and I_m with the same size $[n^{(1)}, n^{(2)}, n^{(3)}]$ in $[0,1]^d$. For simplicity, consider $n^{(1)} = n^{(2)} = n^{(3)} = n$. Since our objective function (1.41) is formed of two terms, we deal with each one separately and sum after. We have:

$$E_D^{SSD} = \frac{1}{2}\int_\Omega (I_m[\Phi(x)] - I_f(x))^2 dx. \tag{1.68}$$

Given a cell-centered grid of width $h = \prod_{l=1}^d h^l$, one gets the discrete measure

$$E_D^{SSD} = \frac{1}{2} h \|I_m[\Phi(x)] - I_f(x)\|^2_{\mathbb{R}^{Nd}}, \tag{1.69}$$

where

$$x = [x_1^{(1)}, ..., x_N^{(1)}, x_1^{(2)}, ..., x_N^{(2)}, ..., x_1^{(d)}, ..., x_N^{(d)}]^T \in \mathbb{R}^{Nd}$$

and the discretization of $y = \Phi(x)$ writes

$$y = [y_1^{(1)}, ..., y_N^{(1)}, y_1^{(2)}, ..., y_N^{(2)}, ..., y_1^{(d)}, ..., y_N^{(d)}]^T \in \mathbb{R}^{Nd}.$$

In this context, $y_j^{(k)} = (\Phi_j^{ref,(k)} - u_j^{(k)}), \quad j = 1 : N, \ k = 1 : d$. In general, $\Phi^{ref}(x) = id(x)$, but when a pre-registration has been used, Φ^{ref} is the result from this pre-registration. Anyway, since Φ^{ref} does not change during registration, one may calculate u and update Φ after each iteration by setting $\Phi = \Phi^{ref} - u$.

Let us set $r \overset{\text{def}}{=} (I_m[x - u(x)] - I_f(x))$, the equation (1.69) can be written

$$E_D^{SSD} = \frac{1}{2} h r^T r, \tag{1.70}$$

and derivatives of this distance can be approximated by:

$$dE_D^{SSD} = hrdr, \tag{1.71}$$

where $dr = dI_m$, and dI_m is the derivatives of the moving image (see (1.38) for dI_m), leading to the Jacobian matrix. The second-order derivatives are approximated from this first-order information because it would be hard to compute exact second derivatives. The approximation is given by

$$d^2E_D^{SSD} = c^{ste}dr^T dr = c^{ste}\mathcal{F}^T\mathcal{F} \stackrel{\text{def}}{=} H_{E_D}, \tag{1.72}$$

that uses the Gauss-Newton method explained in the following chapter. Note that, the matrix (1.72) is symmetric by construction and positive semidefinite. It is positive definite when the Jacobian matrix has full rank.

For more general distance measures (for example Normalized Gradient Fields (NGF)), one gets:

$$\begin{aligned} E_D^{NGF} &= \psi(r(u)), \\ dE_D^{NGF} &= d\psi(r(u))dr(u), \quad \text{and} \\ d^2E_D^{NGF} &= dr(u)^T d^2\psi(r(u))dr(u), \end{aligned}$$

where ψ is some function enabling the distance to be expressed in term of the residual r. For SSD, ψ is a constant. In the FAIR package, a function based on a *midpoint quadrature rule* is provided and returns the current: E_D, dE_D, r,dr and eventually $d^2\psi$. This is returned given inputs $I_m[x - u(x)]$, $I_f(x)$, a cell-centered grid x of N points and the domain Ω. At the same time, the approximation of the hessian $H_{E_D} = d^2E_D = dr(u)^T d^2\psi(r(u))dr(u)$ is done (see [10, p.111]).

For the regularization term, a differential operator B is considered discrete. It is either the discretization of (1.66), its particular case (1.62) or one may use any Laplace-like operator. Thus, discrete regularization writes

$$E_R = \frac{\lambda}{2}h\|Bu\|^2_{\mathbb{R}^{Nd}}, \tag{1.73}$$

where $h = \prod_{l=1}^{d} h^{(l)}$. Hence, we can determine the first derivatives with respect to u:

$$dE_R = \lambda h B^T Bu. \tag{1.74}$$

As opposed to second derivatives of the first term (1.72), here the exact second derivatives are available:

$$d^2E_R = \lambda h B^T B \stackrel{\text{def}}{=} H_{E_R} \tag{1.75}$$

It is important to note that this matrix is symmetric and positive definite by nature. Taking this into account, we form our discrete objective function:

$$\mathcal{F}[u] = E_D^{SSD} + E_R = \frac{1}{2}h\|r\|^2_{\mathbb{R}^{Nd}} + \frac{\lambda}{2}h\|Bu\|^2_{\mathbb{R}^{Nd}}. \tag{1.76}$$

This is a functional since the variable u is a function. Its derivatives are:

$$d\mathcal{F}[u] = dE_D^{SSD} + dE_R = hrdr + \lambda hB^TBu, \tag{1.77}$$

that is the gradient of our functional and its Hessian is approximated by:

$$d^2\mathcal{F}[u] = d^2E_D^{SSD} + d^2E_R = H_{E_D} + H_{E_R} \stackrel{\text{def}}{=} H. \tag{1.78}$$

Observe that, while H_{E_D} is simply semi-positive definite, H_{E_R} is positive definite. The sum is then positive definite and of course symmetric. This gives rise to systems that are symmetric and positive definite.

Computation of these discrete operators are achieved using *finite differences methods* on cell-centered grids. For the first-order operators, the $1D$ derivatives approximation operator may be used, that is

$$\partial_n^{h^k} \approx \frac{1}{h^k}\begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in R^{n^k-1,n^k}. \tag{1.79}$$

For the second-order operators, the Central Finite Differences are performed on a cell-centered grid supposing Dirichlet conditions, that is,

$$\partial_n^{2,h^k} = \frac{1}{(h^k)^2}\begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & -2 & 1 \\ & & 1 & -2 \end{bmatrix} \in \mathbb{R}^{n^k\times n^k}. \tag{1.80}$$

Extension to high dimensions is achieved by Kronecker product. These operators and the obtained linear systems will be addressed in chapter 4.

Optimization

For optimization purposes, we refer to Taylor's theorem to approximate the objective function by its quadratic form at the current point u. Given a function v such that $\|v\|$ is small, Taylor's theorem allows us to write

$$\mathcal{F}[u+v] \approx \mathcal{F}[u] + v^Td\mathcal{F}[u] + v^Td^2\mathcal{F}[u]v. \tag{1.81}$$

Optimizing this function with respect to v leads to $d^2\mathcal{F}[u]v = -d\mathcal{F}[u]$. Then

$$Hv = -d\mathcal{F}[u], \tag{1.82}$$

(where H is given by (1.78)) is the system to be solved to get a new approximation of u that reduces the functional value. Thus, one updates the transformation u by

$$u_{k+1} = u_k + tv, \tag{1.83}$$

where t should verify the Armijo conditions that we explain in chapter 2.

The system (1.82) may be performed using a Preconditioned Conjugate Gradient (PCG) based method with Jacobi, Gauss-Seidel or incomplete Cholesky based preconditioning. For large matrices, Gauss Seidel and Jacobi are more used in FAIR. However, these methods become very time and memory consuming for very large matrices. Accelerating the computation of the system (1.82) will be the topic of the third chapter of this thesis.

The diagram 1.9 illustrates the process in the optimization loop of a typical FAIR algorithm.

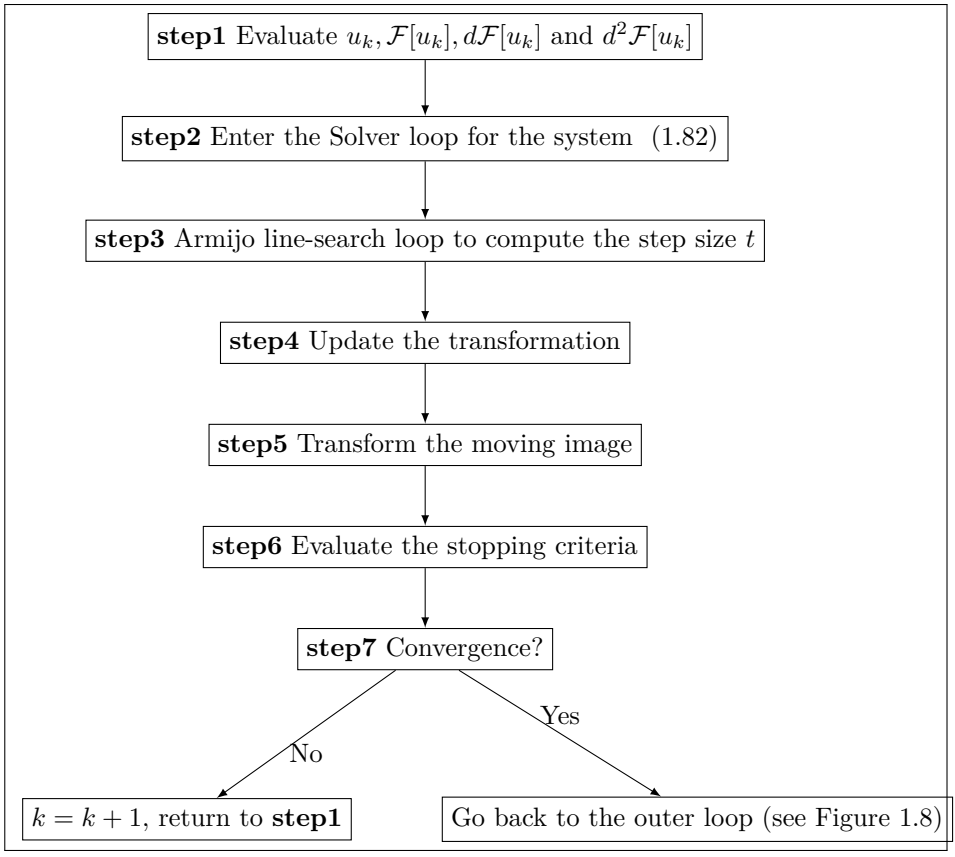


Figure 1.9: FAIR Optimization process (iteration k)

In the algorithms presented in the FAIR package, the most expensive step in the deformable registration process is the resolution of the linear systems. Algorithms that escape to solve linear systems but use convolution after each approximation include the Demons algorithms we present below.

1.4.2 The Demons algorithms

The principle

The main principle of the Demons [10] algorithm can be briefly described as follows. The fixed image is assumed to be in a fix grid while the moving image is in a deformable grid. The deformations of the deformable grid follow the principle of Maxwell's demons. Therefore, based on the polarity principle, the image registration is considered here as a diffusion process where each point [11] in the fixed image is labelled by a "+" ('plus') if its intensity is greater than the intensity of its corresponding point in the fixed image, or by a "−" ('mines') in the opposite case. There is no deformation (diffusion) when intensities are equal at corresponding points.

The following step is to compute elementary forces (attraction forces) that have to deform each pixel/voxel in the moving image, following a descent direction while the orientation of this direction will depend on the label at each point. The steepest descent direction is often chosen by default. Then follows a regularization by convolution using a Gaussian filter. This process is applied in an iterative optimization process that provides an optimal transformation $\hat{\Phi}$ after a sequence of intermediate transformations $\Phi_0, \Phi_1, ..., \Phi_n = \hat{\Phi} \in \mathcal{T}$ ($\mathcal{T}$ being the set of transformations verifying a fixed regularity). In general, the initial transformation is chosen to be the identity transformation $\Phi_0 = Id$.

Formally, consider I_f and I_m, respectively the fixed and the moving images, on a common domain Ω. After each iteration, the deformed image $\Phi_{k+1}(I_m)$ from $\Phi_k(I_m)$ is the result of interactions between $\Phi_k(I_m)$ and I_f created by elementary forces guided by a set of demons D_s.

Informations related to each demon

Informations related to each demon include: the spatial position x in the domain of the image, the current displacement $u(x) = xx'$ where $x' = \Phi^{-1}(x)$ [12] is the new position after deformation of the grid carrying the moving image, the intensity value $I_f(x)$ at the position x, a direction p (in general based on the gradient $\nabla I_f(x)$ of the fixed image) and an orientation "in/out" based, in

10. Here we focus on the algorithm version from [4].
11. Points are used here but it can be a contour or a region as claimed in the indicated reference. In this cas, some threshold are fixed for labelling.
12. At the initial step, $x' = x$.

general, on the difference between intensities in the two images at corresponding positions.

Different variants are obtained by the choice of the positions of the demons (all the points of the grid, certain points of the grid, contour points, etc.), of the set of transformations (see subsection 1.3.8), of the interpolation method (linear, by splines,by sinc, etc.), of the computation of the forces F (based on the gradient, constant magnitude, optical flow like, etc.).

The algorithm Demons 1 framework

The Demons' algorithms *Demons* 0 to *Demons* 3 from [4] follow, with a few differences, the framework described below for *Demons* 1.

Let us denote $I_f(x)$ (respectively $I_m(x)$) the intensity at the position x in the fixed image (respectively in the moving image). First, consider the whole grid of the fixed image as the domain of demons and place demons at each voxel (pixel) of I_f where $\nabla I_f \neq 0$, i.e, $D_s = I_f$ [13]. Then, fix the transformations to be in the set of free form deformations regularized by a Gaussian filter [14]. The following step is to compute the direction $\overrightarrow{d}$ (based on the gradient using only $I_f(x), I_m(x')$ and $\nabla I_f(x)$. Here, $\overrightarrow{d}$ is deduced from the optical flow theory (see [4]) and normalized for numerical stability. The computed value is the elementary deformation

$$\overrightarrow{v} = \frac{(I_m(x') - I_f(x)) \times \overrightarrow{\nabla} I_f(x)}{(I_m(x') - I_f(x))^2 + \|\overrightarrow{\nabla} I_f(x)\|^2}, \tag{1.84}$$

that is considered here as a displacement and then the direction is $\overrightarrow{d} = -\overrightarrow{v}$. In what follow, we set $v = \overrightarrow{v}$ omitting the arrow over v for simplicity.

To move in the domain, domain points are labelled following these principles: one may either fix a constant threshold c in the fixed image and then decide to labell "+" (Out) if $I_f(x') > c$ at each point x' and "-" (In) if $I_f(x') < c$. One can also, as in Demons 1, label "+"(Out) if $I_m(x) - I_f(x') > 0$ and "-" (In) if $I_m(x) - I_f(x') < 0$.

In Demons 1, v is considered as an elementary deformation, i.e, $v = \delta\Phi$. Setting

$$x'_0 = \Phi_0^{-1}(x) = x,$$

the transformation is then updated by

$$\Phi_{k+1} = \Phi_k + \delta\Phi_k, \quad k = 0, 1, \ldots.$$

13. In (*Demons* 0 and *Demons* 2, demons are considered only at contours points while in *Demons* 3 demons are placed between two adjacent voxels of different labels.

14. Rigid transformation for *Demons* 0, affine for *Demons* 2 and *Demons* 3.

The displacement $\delta\Phi_k$ is regularized by convolution with a Gaussian filter while each position x'_k is updated by computing the following position

$$x'_{k+1} = \Phi^{-1}_{k+1}(x).$$

At the new position, the intensity $I_m(x')$ is estimated using trilinear (or bilinear for 2D images) interpolation.

According to Thirion [4], v is an elementary deformation and not a velocity. In some later variants, most of time v is considered as a velocity field. Furthermore, the addition in $\Phi_{k+1} = \Phi_k + \delta\Phi_k$, does not allow the transformation Φ_{k+1} to preserve properties of the elementary displacements $\delta\Phi_k$ since it does not provide a group structure to the set of functions.

The variant proposed by [37] overcomes the difficulty by using the composition $\Phi_{k+1} = \Phi_k \circ \delta\Phi_k$, and obtains invertible transformation. In addition, he proposed an other variant in [38] called "diffeomorphic demons", where the transformation is diffeomorphic. In this variant, v is a velocity field that has to be integrated to obtain a displacement field.

Apart from a few variants, these algorithms have the advantage that the computation of the gradient is made only once before the iterative process because it is calculated within the fixed image. In addition, the pixels or voxels of the moving image are simultaneously stretched or narrowed according to the labels. This allows a significant gain in computation time. There are many variants of the Demons algorithms (see [1],[4],[38] and [39] but the most popular are *Symmetric and Diffeomorphic Demons*.

However, in spite of the success of these algorithms, the linear complexity they offer remains relatively expensive (in terms of computation time and storage) with respect to some clinical applications, particularly for high resolution or high dimensional images.

Chapter 2

Introduction to nonlinear optimization methods

2.1 Fundamentals of unconstrained optimization problems

Optimization is a process of finding the best solution in a specified environment or under defined circumstances. This practice can be expressed as a search for conditions that minimize effort or maximize benefit. To address the problem, the effort or benefit is modelled as a *function of some variables*. The nature of such a function, the information available on that function or its variables, and the nature of the variables, lead to different types of optimization problems [40]. In addition, these variables may be subject to constraints that the solution has to verify. These constraints are usually expressed as equalities or inequalities. This gives rise to constrained and unconstrained optimization problems.

There is no method designed to efficiently solve all optimization problems. Instead, different methods have been developed to address specific optimization problems. In this thesis, the function is considered real and nonlinear, the variables are real numbers and no constraints are assumed. Since it is known that in applications, variables are often subject to constraints, unconstrained optimization techniques may be considered to be less important. However, these techniques are used for more general problems, including reformulated constrained and global minimization problems. This chapter is mainly inspired from the book of J. Nocedal and S.J. Wright [40].

A general, unconstrained nonlinear optimization problem is defined as

$$\min_{x \in \mathbb{R}^n} f(x), \tag{2.1}$$

where the minimization formulation is commonly used since any maximization problem can be converted into a minimization problem due to the equivalence $max(f) = -min(-f)$. The real function $f : \mathbb{R}^n \to \mathbb{R}$ denotes a continuous function and is mostly called *objective function* in the optimization field but it is also known as *cost function*, *loss function*, *utility function* or *energy functional* in other fields [41].

The optimization process consists of choosing values of x in the defined domain that leads to lower value of f. A *global minimizer* of problem (2.1) is a vector $x^* \in \mathbb{R}^n$ satisfying $f(x^*) \leq f(x), \quad \forall\, x \in \mathbb{R}^n$. In general, it is difficult to find the global minimizer of f for nonconvex and nonlinear functions. Instead, one is interested in identifying a point x^* achieving the smallest value of f in a neighborhood $\mathcal{N} \subset \mathbb{R}^n$. Such a solution is known as *local solution*. A *local minimizer* x^* is a point such that $f(x^*) \leq f(x), \forall\, x \in \mathcal{N} \subset \mathbb{R}^n$. Such a solution is known to be weak since it may not be unique. Then, a *strict local minimizer* x^* is a point such that $f(x^*) < f(x),\ \forall\, x \in \mathcal{N} \subset \mathbb{R}^n,\ x \neq x^*$.

Searching for local minimizer is of interest. Firstly because it can be processed iteratively to approximate a global minimizer. Secondly, it can be sufficient for the decision maker and thirdly, it can be the global minimizer under certain conditions. The theorem below states the conditions under which a local minimizer is also the global minimizer.

Theorem 2.1

When f is a convex function, any local minimizer x^* is a global minimizer of f. If in addition f is differentiable, then any stationary point, that is a point x such that $\nabla f(x) = 0$, is a global minimizer of f.

Proof. See [40], p. 16 □

Although the case in theorem 2.1 seems simpler, it is the base for many iterative methods. For example, when the objective function is differentiable, one may iteratively minimize its quadratic approximation. When this quadratic approximation being is convex, Theorem 2.1 is applied. This is repeated until certain criteria are met. These techniques are studied in the following subsection.

2.1.1 Optimality conditions

To conclude that a given point x^* is a local minimizer, instead of examining all the points in the neighborhood of x^*, one may use theoretical results to identify the local minimizer when the function is sufficiently smooth. These results are generally derived from the Taylor's theorem. While there are many versions of Taylor's theorem, here we are interested in the one that states how

and under which conditions, a general smooth nonlinear function is well approximated by a linear or a quadratic function at a local point. Then, we will focus on methods that use the gradient ∇f and the Hessian $\nabla^2 f$ to minimize the function f.

Theorem 2.2 (*Taylor's theorem*)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuoustly diffenrentiable and that $p \in \mathbb{R}^n$. Then we have that

$$f(x+p) = f(x) + \nabla f(x+tp)^T p, \tag{2.2}$$

for some $t \in [0,1]$. Moreover, when f is twice continuously differentiable, we have that

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp) p dt, \tag{2.3}$$

and that

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x+tp) p, \tag{2.4}$$

for some $t \in [0,1]$.

Proof. See for example [42, p. 98] □

The three following theorems state the optimality conditions for problem (2.1) and their proofs are available in [40]. The theorem below, known as the first-order optimality condition, assumes that x^* is a local minimizer, then deduces a property about the gradient $\nabla f(x^*)$. This is important because, if one knows how the gradient behaves at a local minimizer, then one may be looking at the gradient behaviour to characterize candidates to optimality.

Theorem 2.3 (*First-order necessary optimality condition*)

If x^* is a local minimizer, and f is continuously differentiable in an open neighborhood of x^*, then

$$\nabla f(x^*) = 0. \tag{2.5}$$

Proof. See [40, p.15] □

The points that verify (2.5) are called *first-order stationary points*. A stationary point might not be a solution although many algorithms are designed simply to determine stationary points. A second-order necessary condition has been established using second-order derivatives information in addition to the gradient information. It is stated in the following theorem.

Theorem 2.4 (*Second-order necessary optimality conditions*)

If x^* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^*, then

$$\nabla f(x^*) = 0 \quad \textit{and} \quad \nabla^2 f(x^*) \quad \textit{is positive semi definite.} \tag{2.6}$$

Proof. See [40, p.15] □

A guarantee that x^* is a strict local minimizer of f is obtained when we have a sufficient condition on the derivatives of f at the point x^*. This is stated is the following theorem.

Theorem 2.5 (*Second-order sufficient optimality conditions*)

If $\nabla^2 f$ is continuous in an open neighborhood of x^*, $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, Then x^* is a strict local minimizer of f.

Proof. See [40, p.16] □

This theorem gives the characteristics that ensure that one gets a local minimizer. Thus, whenever the gradient goes to zero and the Hessian is positive definite at a point y, any small move from y will increase the function value. We conclude that we are at a local minimizer $y = x^*$.

In general, an algorithm for unconstrained minimization problem generates a sequence of iterates $\{x_k\}_{k=0,1,2,\ldots}$ where the user supplies a starting point x_0. The choice of x_0 may be guided by the knowledge of the application at hand, by a systematic approach or in some arbitrary manner. Starting at x_0, the algorithm will terminate when either no more progress is possible, or when it seems that a solution has been approximated with sufficient accuracy. To move from one iterate x_k to the next iterate x_{k+1}, the algorithm uses informations about the function at x_k and possibly at previous iterates, while imposing the function to a lower value at x_{k+1} than at x_k [1]. In moving from x_k to x_{k+1} it is common to look for a *direction* p_k in which the condition $f(x_{k+1}) < f(x_k)$ is the most satisfied and *a distance* of the move to this next iterate.

Two fundamental strategies exist for this move (from x_k to x_{k+1}). The *trust-region strategy* and the *line-search strategy*. They differ in the order in which they choose the direction and the distance of the move. The line-search strategy first fixes the direction p_k, then identifies an appropriate distance α_k known as the *step length*. The trust-region strategy on the other hand, first chooses a maximum distance Δ_k, namely *the trust-region radius*, and then searches for a *direction* and a *step-length* that improves better, subject to this

1. There exist nonmonotone algorithms that do not insist on a decrease in f at every step but after some prescribed m iterations.

distance constraint. In this work, only the principles of the trust-region strategy will be recalled, while the focus is on the line-search strategy.

In the remain, we use the notation $f_k, \nabla f_k, \nabla^2 f_k$ to denote respectively the function value $f(x_k)$, the gradient $\nabla f(x_k)$ and the Hessian $\nabla^2 f(x_k)$ of the function at x_k.

2.1.2 Line-search strategy

In line-search strategy, new points are computed from previous ones, following iterates of the type

$$x_{k+1} = x_k + \alpha_k p_k. \tag{2.7}$$

Thus, to update the current value, one needs to know the *search direction* p_k that informs in which direction the function decreases and a scalar α_k called the step length, that specifies how far one should go in that direction to obtain a sufficient decrease in the function.

Descent directions

Consider a vector $x_k \in \mathbb{R}^n$, a *descent direction* p_k at x_k is such that

$$p_k^T \nabla f_k < 0. \tag{2.8}$$

Line-search methods use any descent direction, one that guaranties a decrease in f. A descent direction usually takes the form

$$p_k = -B_k^{-1} \nabla f_k. \tag{2.9}$$

It can be shown that each direction that makes strictly less than $\pi/2$ randian angle with $-\nabla f_k$ produces a decrease in the value of f, provided that the step length is sufficiently small.

To verify this, observe that according to Taylor's theorem, for any direction p_k and any small positive scalar (step length) α_k, we have

$$f(x_k + \alpha_k p_k) = f_k + \alpha_k p_k^T \nabla f_k + \mathcal{O}(\alpha_k^2 \|p_k\|^2). \tag{2.10}$$

The rate of change in f along the direction p_k at the current point x_k is the coefficient of α_k, namely $p_k^T \nabla f_k$. Hence, decrease is guaranteed with the unit direction p_k, that is the solution of the problem

$$\min_{p_k} \; p_k^T \nabla f_k, \text{ subject to } \|p_k\| = 1. \tag{2.11}$$

Consider θ the angle between p_k and ∇f_k, we have $p_k^T \nabla f_k = \|p_k\| \|\nabla f_k\| \cos\theta$. Since $-1 \leq cos(\theta) \leq 1$, the minimizer is attained when $\cos\theta = -1$ and then

$$p_k = -\nabla f_k / \|\nabla f_k\| \tag{2.12}$$

offers the most obvious choice of search direction with sufficient decrease. This direction is called *steepest descent direction* and is orthogonal to the contours of the function.

The *steepest descent method* is a line-search method that uses $p_k = -\nabla f_k$ at each step. Easy computation is the main advantage of the steepest descent method, since it requires only first derivatives of the function. In addition, the convergence of the steepest descent method is guaranteed. However it is known to be slow, especially for ill-scaled problems.

Instead of using the first-order Taylor's series (2.10), we may use the second-order approximation. This gives:

$$f(x_k + \alpha_k p_k) \approx f_k + \alpha_k p_k^T \nabla f_k + \frac{1}{2}\alpha_k^2 p_k^T \nabla^2 f_k p_k \stackrel{def}{=} m_k(p_k). \tag{2.13}$$

Assuming that $\nabla^2 f_k$ is positive definite, the minimizer of $m_k(p_k)$ is called the *Newton direction* that is the vector p_k^N, given by

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k. \tag{2.14}$$

When the model $m_k(p_k)$ approximate sufficiently well the true function $f(x)$ near the current point x_k, the Newton direction is interesting because it catches the curvature of the quadratic model. Unless the gradient ∇f_k (and consequently p_k^N) is zero, we have that $\nabla f_k{}^T p_k^N = -\nabla f_k{}^T (\nabla^2 f_k)^{-1} \nabla f_k < 0$ because $(\nabla^2 f_k)^{-1}$ is positive definite. This indicates that the Newton direction is a descent direction. However, if $\nabla^2 f_k$ is not positive definite, the Newton direction may be not defined because $(\nabla^2 f_k)^{-1}$ may not exist and when it exists, it is not guaranteed that it is a descent direction since $\nabla f_k{}^T (\nabla^2 f_k)^{-1} \nabla f_k$ is not always positive.

Newton methods are optimization methods that use the newton direction. Their main advantage is that, they have a quadratic rate of convergence especially when they are in the neighbourhood of the solution. The main disadvantange of this direction is the need for explicit computation of second derivatives $\nabla^2 f_k$ constituting the Hessian that are often expensive, unmanageable and error-prone. An alternative to Newton's direction is the so called *quasi-Newton direction*, where instead of computing the true Hessian, one uses an approximation $B_k \approx \nabla^2 f_k$. We have

$$p_k^{QN} = -B_k^{-1} \nabla f_k, \tag{2.15}$$

and this direction is often updated at each iteration. This direction do not require an explicit computation of the Hessian but if B_k is positive definite, quasi-Newton methods may still attain a local superlinear convergence rate. In practice, the fact that the changes in the gradient provide informations about

second derivatives along the search direction is exploited. That is (see [40, p.137])

$$B_k \alpha_k P_k^{QN} \approx \nabla f_{k+1} - \nabla f_k. \tag{2.16}$$

More practical use of these methods approximate at each iteration the inverse of B_k instead of B_k itself. This makes the quasi-Newton methods very attractive for large-scale optimization problems where it is expensive to explicitly compute the Hessian due to the high number of variables.

Step length

In line-search strategies, when the direction is chosen, the question is *how long should we go in that direction*. There are two possibilities, either one determines the exact step length or one can use some heuristics. The exact line search aims to determine the step length α_k that minimizes exactly $\phi(\alpha_k) = f(x_k + \alpha_k p_k) \approx f_k + \alpha_k p_k^T \nabla f_k + \alpha_k^2 p_k^T \nabla^2 f_k p_k,\ \alpha_k > 0$. However, in general, this exact line search is impossible or too expensive. The alternative is the use of inexact line searches that aim to achieve sufficient decrease in the function f using some heuristics.

To allow sufficient decrease in the objective function, the step length α_k has to verify the inequality

$$f(x_k + \alpha_k p_k) \leq f_k + c_1 \alpha_k \nabla f_k^T p_k, \tag{2.17}$$

for some constant $c_1 \in [0, 1]$. This inequality (2.17) is also called *Armijo condition*. The scalar c_1 is chosen in practice to be small, say $c_1 = 10^{-4}$. The sufficient decrease condition is not always enough to make reasonable progress. It can admit a very small step length. To avoid all these very short steps, a second condition known as *curvature condition* has to be satisfied. This requires

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k, \tag{2.18}$$

for some constant $c_2 \in [c_1, 1]$, where c_1 is the constant from (2.17). The curvature condition ensures that the slope of ϕ, $\phi'(\alpha)$ at α_k is greater than c_2 times the initial slope $\phi'(0)$. This indicates for a slope $\phi'(\alpha)$ strongly negative that moving further along the choosen direction one can reduce f significantly. The sufficient decrease condition and the curvature condition are known collectively as the *Wolfe conditions*.

Another heuristic is the Goldstein conditions that ensure that the step length α achieves sufficient decrease but is not too short. It has to verify the double inequalities

$$f(x_k) + (1 - c)\alpha_k \nabla f_k^T p_k \leq f(x_k + \alpha_k p_k)^T p_k \leq f(x_k) + c\alpha_k \nabla f_k^T p_k, \tag{2.19}$$

with $0 < c < 1/2$. The Goldstein conditions are often used in Newton-type methods but are not well suited for quasi-Newton methods that maintain a

positive definite Hessian approximation. Many algorithms do not explictly evaluate all the conditions. They use instead the so-called *backtraking strategy*. The idea of backtracking is to skip the test for the curvature condition and use only the sufficient decrease test. For this purpose, an initial value α_0 is set sufficiently high and is reduced iteratively until the sufficient decrease (2.17) is verified. Here is a pseudo code of the backtracking algorithm.

Algorithm 2.1 (*Backtracking Line Search*)

Choose $\alpha_0 > 0,\ c \in [0\ 1],\ \rho \in [0\ 1];\ \alpha \leftarrow \alpha_0;$
  While $f(x_p + \alpha p_k) > f(x_k) + c\alpha \nabla f(x_k)^T p_k$
    $\alpha \leftarrow \rho\alpha;$
  end*(While)*

On the one hand, the initial trial step length is often taken to be $\alpha_0 = 1$ for Newton and quasi-Newton directions. This choice ensures that unit step lengths will always be taken whenever the algorithm is in a neighbourhood of the solution, and this allows the high rate-of-convergence of these methods to take effects. On the other hand, methods that do not provide well scaled search directions, such as the steepest descent method and conjugate gradient methods, should use information on the algorithm at the current and previous iterations to make an initial guess.

One of the most used strategies is to interpolate data within $f_{k-1},\ f_k$ and $\nabla f_{k-1}^T p_k$. One of the most used formulae is:

$$\alpha_0 = \frac{2(f_k - f_{k-1})}{\nabla f_k^T p_k}. \tag{2.20}$$

This initial step length is used in the presented algorithm in chapter 5.4.

Trust-region strategy

The trust-region strategy is an alternative to the line-search strategy. This method uses information gathered about f, to construct a model function m_k whose behaviour near the current point x_k is similar to that of f. This model m_k is then minimized in a well designed region around x_k where it is expected that it approximates well the objective function f. This means, at each iteration we approximately solve the *trust-region subproblem:*

$$\min_{p \in \mathbb{R}^n} m_k(x_k + p), \quad \text{where} \quad x_k + p \quad \text{lies inside the trust-region.} \tag{2.21}$$

In most cases, the trust-region is a ball defined by $\|p\|_2 < \Delta$ where Δ, the *trust-region radius*, is a strict positive scalar. When the candidate solution does not produce a sufficient decrease in f, it would be due to the fact that the region is too large, then the region may be shrunk and (2.21) is re-resolved. In other cases, either the region is kept identical or it is enlarged. For details on this strategy see [40] and [43].

2.2 Conjugate Gradient (CG) methods and linear systems

Problems with very large size, say millions of variables, can be solved efficiently only when the computational cost and the storage requirements are kept at a tolerable level. Some approaches are more effective for these problems within the Conjugate Gradient (CG) method. Other methods include, appropriate inexact Newton methods (based on sparse factorizations of the Hessian matrix when factorization methods are affordable) and adapted quasi-Newton methods (e.g. those exploiting structural properties such as *partial separability*).

Now, let us consider a specific case of a minimization problem (2.1) where the objective function is a *quadratic function* given by

$$q(x) = \frac{1}{2}x^T Ax - b^T x, \tag{2.22}$$

with $A \in \mathbb{R}^{N\times N}$ and $b \in \mathbb{R}^N$. By the second-order necessary optimality conditions (see 2.4), if x^* is a local minimizer of f, then the gradient given by $\nabla q(x^*) = Ax^* - b$ has to be zero and the Hessian,$\nabla^2 q(x^*) = A$, has to be positive definite. In this case, the minimizer x^* is the unique solution of the linear system

$$Ax = b. \tag{2.23}$$

Solving efficiently (2.23) for n very large, is an active research area and is very useful in scientific computing. To introduce the CG method, we define first the Krylov subspace. Consider a matrix $A \in \mathbb{R}^{N\times N}$, the Krylov subspace is the space of all vector $x \in \mathbb{R}^N$ that can be written as $x = p(A)v$ where p is a polynomial of degree k and v is an appropriate vector. This is stated in the following definition for the linear system (2.23).

Definition 2.6 (*Krylov subspace*)

Given the matrix $A \in \mathbb{R}^{N\times N}$ in (2.23) and an initial solution $x_0 \in \mathbb{R}^N$, the Krylov subspace of degree $k \leq N$ generated by the matrix A and the vector $r_0 = b - Ax_0$ is defined by

$$\mathcal{K}_k(A, r_0) = span\{r_0, Ar_0, \cdots, A^{k-1}r_0\}. \tag{2.24}$$

The CG method is one of the so-called *Krylov methods*. These methods minimize over a set of nested Krylov subspaces (2.24). They are among the most successful methods currently available in numerical linear algebra[44]. Introduced by Stiefel and Hestenes (1952), the Conjugate Gradient (CG) algorithm is among the most used iterative method for solving large linear systems

with positive definite matrices. In addition, it has been adapted to solve general non-linear optimization problems [40]. Let us now present the general CG method.

2.2.1 The CG method

The CG method is an alternative to Gaussian elimination well suited for *solving large linear systems*. As stated above, The CG method minimizes the quadratic function (2.22) on the Krylov subspace (2.24) assuming $A \in \mathbb{R}^{N\times N}$ is symmetric and positive definite. Its performance is related to the *distribution of the eigenvalues* of the coefficient matrix A. The minimizer of such convex quadratic function is the solution of the linear system (2.23), since the gradient of this convex quadratic function is the residual of the linear system. That is,

$$\nabla q(x) = Ax - b. \tag{2.25}$$

Consider $x = x_k$, then

$$r_k \stackrel{def}{=} b - Ax_k. \tag{2.26}$$

Starting with an initial guess $x_0 \in \mathbb{R}^N$, set $r_0 = b - Ax_0$ and an initial direction (the steepest descent direction) $p_0 = -\nabla q(x_0) = r_0$. Then, the CG algorithm generates a sequence of iterates $\{x_k\}$ where each iterate is computed with small informations from small previous iterates. Here we use the so called *three two-terms recurrence*. This means, the job is done by three vectors that need another term each, to be updated (two terms) in the loop iteration.

To establish the CG algorithm, let us consider x_0 as given and

$$x_{k+1} = x_k + \alpha_k p_k, \quad k = 1, 2, \ldots. \tag{2.27}$$

By minimizing $\phi(\alpha_k) = q(x_k + \alpha_k p_k)$ from (2.22) exactly, we get

$$\alpha_k = \frac{r_k^T p_k}{p_k^T A p_k}. \tag{2.28}$$

This is an exact step length. Observe that at iteration $k+1$, (2.27) in (2.26) gives

$$r_{k+1} = r_k - \alpha_k A p_k, \tag{2.29}$$

that is an efficient fast update of the residual. The CG algorithm economically generates a set of directions with a property known as *A-conjugacy* defined

below.

Definition 2.7 (*A-conjugate directions*)

Nonzeros vectors $\{p_0, p_1, \ldots, p_l\}$ *are said to be* conjugate *with respect to the symmetric positive definite matrix* A *if*

$$p_i^T A p_j = 0, \quad \forall i \neq j. \tag{2.30}$$

Starting by the initial steepest descent direction $p_0 = r_0$, the new directions $p_k,\ k = 1, 2, \ldots$ are computed as a linear combinations of the residuals r_k and the previous direction p_{k-1}. Thus,

$$p_k = r_k + \beta_k p_{k-1}, \tag{2.31}$$

where the scalar β_k has to be determined by the requirement that p_{k-1} and p_k must be conjugate with respect to A. To determine β_k, let us premultiply (2.31) by $p_{k-1}^T A$ while imposing the A-conjugacy $p_{k-1}^T A p_k = 0$, we obtain

$$\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}}. \tag{2.32}$$

We need also the residuals to be orthogonal to the directions and each of them to belong to the Krylov subspace defined above. This ensures that the minimizer is a sum of x_0 and a linear combination of A-conjugate directions. These arguments are stated in the theorem below, whose demonstration can be found in [40].

Theorem 2.8 (*Expanding subspace minimization*)

Let $x_0 \in \mathbb{R}^N$ be any starting point and suppose that the sequence $\{x_k\}$ is generated by the CG algorithm applying (2.27), (2.28). Then

$$r_k^T p_i = 0, \quad \text{for } i = 0, 1, \ldots, k-1, \tag{2.33}$$

and x_k is the minimizer of $q(x) = x^T Ax - b^T x$ over the set

$$\{x | x = x_0 + span\{p_0, p_1, \ldots, p_{k-1}\}\}. \tag{2.34}$$

Proof. See [40, p.106]. □

When the residuals are mutually orthogonal while each of them and each direction belong to the Krylov subspace, the theorem below ensures that the CG method will terminate in a finite number of iterations, at most n. This property can be lost if orthogonality is destroyed.

Theorem 2.9

Suppose that the k^{th} iterate generated by the Conjugate Gradient method is not the solution point x^*. The following properties hold

$$r_k^T r_i = 0, \quad \text{for } i = 0, 1, \ldots, k-1, \tag{2.35}$$

$$span\{r_0, r_1, \ldots, r_k\} = span\{r_0, Ar_0, \ldots, A^k r_0\}, \tag{2.36}$$

$$span\{p_0, p_1, \ldots, p_k\} = span\{r_0, Ar_0, \ldots, A^k r_0\}, \tag{2.37}$$

$$p_k^T A p_i = 0, \quad \text{for } i = 0, 1, \ldots, k-1. \tag{2.38}$$

Therefore, the sequence $\{x_k\}$ converge to x^* in at most N steps.

Proof. See [40, p.109]. □ In practice, from (2.33) and (2.31), we compute

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}, \tag{2.39}$$

and using the fact that $\alpha_k A p_k = r_k - r_{k+1}$, we have

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}. \tag{2.40}$$

This enables a simple algorithm to be performed with three two-term recurrences in a complete CG algorithm (see Algorithm (2.2)).

Algorithm 2.2 (*CG algorithm*)

Given x_0;
Set $r_0 \leftarrow b - Ax_0$; $p_0 \leftarrow r_0$; $k \leftarrow 0$;
While $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$
$$r_{k+1} \leftarrow r_k - \alpha_k A p_k;$$
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k};$$
$$p_{k+1} \leftarrow r_{k+1} + \beta_{k+1} p_k;$$
$$k \leftarrow k + 1;$$

end(While)

Note that, at each iteration in Algorithm (2.2), we need no more than the information from the last two iterations. The computational task concerns the matrix-vector product Ap_k and two inner products $p_k^T(Ap_k)$ and $r_{k+1}^T r_{k+1}$.

Let us investigate the amount of work involved in the Algorithm (2.2) in terms of flops [2]. Assuming the matrix A is full, in each loop, three inner products are computed to involve $3 * (2N) = 6Nflops$, one matrix-vector product that involves $2N^2 flops$, three scalar-vector products involve $3N$ flops and three additions that involve $3N$ flops. In total, the amount of flops in each loop is approximated to $N_{_{flops}} = 2N^2 + 12N$.

In finite precision, the CG method may loose the properties above. In that case, the convergence behaviour of the algorithm is affected. This will be addressed in Subsection 4.2.1. In exact arithmetics, the definition below offers an important indicator on how the CG method may behave on a specific linear system. It is called the condition number of the coefficient matrix of the system.

2. Here we consider a flop to be the amount of work associated to a single point floating operation including addition (or substracting) and multiplication.

Definition 2.10 (*Condition number*)

Given a nonsingular matrix A, the condition number of A *is the quantity noted and defined by*

$$k(A) = \|A\|\|A^{-1}\|, \tag{2.41}$$

where any matrix norm can be used. For symmetric positive definite matrices and Euclidean norm, the condition number can be expressed by $k(A) = \dfrac{\lambda_N}{\lambda_1}$ where λ_N and λ_1 are respectively the greatest and smallest eigenvalues of A. In this case, $k(A)$ is the spectral condition number of A.

2.2.2 The CG convergence rate

Theorem 2.9 states that in exact arithmetic, the CG algorithm will terminate at the solution in at most N iterations. It is interesting to note that, if the distribution of eigenvalues of A has favorable features, the algorithm will identify the solution in far fewer than N operations. This property is stated in the following theorems.

Theorem 2.11

If $A \in \mathbb{R}^{N\times N}$ has $r \leq N$ distincts eigenvalues, the CG algorithm 2.2 will terminate with $x_k = x^*$ for some $k \leq r$.

Proof. See [43, p. 85] □

This states that the number of CG iterations does not exceed the rank of A. Clearly, the more the eigenvalues of A are clustered, the faster is the CG algorithm. Note that, given a symmetric positive definite matrix A, the A-norm of a vector x is defined by

$$\|x\|_A = \sqrt{x^T Ax}.$$

Theorem 2.12 below states that reducing the condition number of the matrix A improves the convergence rate of the CG algorithm.

Theorem 2.12

The error $\epsilon_k = x_k - x^*$ of the iterates generated by the CG algorithm 2.2 satisfies the inequality

$$\frac{\|\epsilon_k\|_A}{\|\epsilon_0\|_A} \leq 2\left(\frac{\sqrt{k(A)}-1}{\sqrt{k(A)}+1}\right)^k, \tag{2.42}$$

where $k(A)$ is the spectral condition number of A.

Proof. See [43] p. 85. □

Thus, the following theorem has to be stated.

Theorem 2.13

If A has eigenvalues $\lambda_N \geq \lambda_{N-1} \geq \ldots \geq \lambda_1 > 0$ then we have

$$\frac{\|x_{k+1} - x^*\|_A^2}{\|x_0 - x^*\|_A^2} \leq \left(\frac{\lambda_{N-k} - \lambda_1}{\lambda_{N-k} + \lambda_1}\right)^2 \tag{2.43}$$

Proof. See [40, pp.115-116] and indication therein. □

This theorem states that the convergence rate of the CG algorithm is also affected by the distribution of the smallest eigenvalues. Theorems 2.12,2.11 and 2.13 above are the basis of the idea of *preconditioning* the CG algorithm, presented roughly in the following subsection and developed in chapter 4.

2.2.3 Preconditioned Conjugate Gradient (PCG)

Preconditioning techniques aim to transform a system $Ax = b$ into a new equivalent system with a more favourable eigenvalues distribution. To accelerate Krylov subspace methods, one may act on the spectral properties of the matrix since this impacts their convergence rate. In particular, since the CG method requires only matrix-vector products at each iteration (and few vector operations such as dot products and vector updates), the major computational work depends on the number of iterations it may take to obtain an acceptable level of accuracy.

Although for sparse or structured matrices the CG method may be efficiently implemented, it requires preconditioning to be really effective [45] and ensure that the number of iterations is kept relatively small. In practice, if the matrix A is symmetric and positive definite, which must be the case for CG algorithms, the required *preconditioner* is a non singular matrix P that has to be symmetric, positive definite and such that the condition number of $P^{-1}A$, $k(P^{-1}A)$, is decreased compared to $k(A)$. When this preconditioner is available or has been designed implicitly, the Preconditioned Conjugate Gradient (PCG) algorithm is used. The PCG algorithm from [40, P. 119] is given in

Algorithm 2.3 below.

Algorithm 2.3 (*PCG algorithm*)

Given x_0;
Set $r_0 \leftarrow Ax_0 - b$;
Solve $Py_0 = r_0$ *for* y_0;
Set $p_0 \leftarrow -y_0$, $k \leftarrow 0$;
While $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T y_k}{p_k^T A p_k};$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k;$$
$$r_{k+1} \leftarrow r_k + \alpha_k A p_k;$$
Solve $Py_{k+1} = r_{k+1}$;
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T y_{k+1}}{r_k^T y_k};$$
$$p_{k+1} \leftarrow -y_{k+1} + \beta_{k+1} p_k;$$
$$k \leftarrow k + 1;$$
end(While)

Techniques to design effective preconditioners are discussed in chapter 4.

2.3 Nonlinear least-squares problems

Least-squares problems occur in many applications such as data fitting, prediction and model calibration. This is also the case for MIRP (see (1.65)). The minimization formulation of a least-squares problem has the special form

$$\min_{x \in \mathbb{R}^N} f(x) = \min_{x \in \mathbb{R}^N} \frac{1}{2} \sum_{j=1}^{m} r_j^2(x), \tag{2.44}$$

where each $r_j : \mathbb{R}^N \to \mathbb{R},\ j = 1 : m$ is referred to as the *residual*. To minimize the objective function (2.44) efficiently, we have to exploit its special structure and its derivatives. Gathering the individual residual r_j in a residual vector

$$r : \mathbb{R}^N \to \mathbb{R}^m,$$

such that

$$r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^T,$$

we can rewrite (2.44) as

$$\min_{x \in \mathbb{R}^N} f(x) = \min_{x \in \mathbb{R}^N} \frac{1}{2} \|r(x)\|_2^2 = \min_{x \in \mathbb{R}^N} \frac{1}{2} r(x)^T r(x). \tag{2.45}$$

The task is to determine the vector $x \in \mathbb{R}^N$ that minimizes the 2-norm of the function (2.44) (see [46]). When the residual functions r_j are all linear, the problem is a linear least-squares. But in our case, the functions r_j in (2.25) are nonlinear. The idea is to replace the function (2.44) at the current point by its quadratic Taylor's approximation. However, this quadratic approximation needs second derivatives of each residual function, and this makes the strategy really prohibitive. Instead of computing the exact Hessian, these second derivatives are approximated to constitute the well known *Gauss-Newton method* (see [40] or [46]) briefly presented below.

2.3.1 Gauss-Newton Method

Let us consider that the m residual functions are smooth and continuously differentiable. The gradient of the function (2.44) is:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x) \tag{2.46}$$

where $J(x) = \begin{pmatrix} \nabla r_1(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{pmatrix} = \begin{pmatrix} \frac{\partial r_1}{\partial_{x_1}} & \cdots & \frac{\partial r_1}{\partial_{x_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial_{x_1}} & \cdots & \frac{\partial r_m}{\partial_{x_n}}. \end{pmatrix}$ is the Jacobian matrix.

The Hessian is given by:

$$\nabla^2 f(x) = \sum_{j=1}^{m} \nabla r_j(x)\nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x),$$

that can be written using Jacobian notations as

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x). \tag{2.47}$$

However, the computation of the right-hand second term is expensive for large problems. Thus, assuming either the residuals are close to affine functions near the solution or residuals are really small enough (i.e $\|\nabla^2 r_j(x)\|$ are small), the second term in (2.47) is relatively small compared to the first one and can be neglected. Such an approximation is often effective especially when the term $J(x)^T J(x)$ dominates the second term in (2.47). In this case, the individual norms of second derivatives of the residuals (that is, $|r_j(x)|\|\nabla^2 r_j(x)\|$ are sufficiently small compared to eigenvalues of the approximation $J(x)^T J(x)$. When the approximation is really effective, the convergence rate of the Gauss-Newton method is rapid (near quadratic). However, when these assumptions are not verified, the approximation is not good and the convergence rate is linear. In

this case, it is better to use quasi-Newton methods or to refer to hybrid methods [40, page 262].

The major trick for least-squares problems minimization is that the Hessian is approximated by Jacobian informations at x_k:

$$H_k = H(x_k) \stackrel{\text{def}}{=} J(x_k)^T J(x_k). \tag{2.48}$$

This approximated Hessian is symmetric, however it is positive definite only if the Jacobian is of full-rank. When minimizing the function (2.44), we can use the line-search strategy where the search direction p_k^{GN} is the solution of the linear system

$$J(x_k)^T J(x_k) p_k^{GN} = -J(x_k)^T r(x_k). \tag{2.49}$$

This equation known as *normal equation* can be written

$$H p_k = g \tag{2.50}$$

where $H = J(x_k)^T J(x_k)$ and $g = -J(x_k)^T r(x_k)$. Another advantage of the Gauss-Newton method is the fact that, for a full-rank Jacobian and non-zero gradient $\nabla f_k = -J(x_k)^T r(x_k)$, the search direction p_k^{GN} is always a descent direction. This ensures effectiveness of a line search strategy. Here is a pseudo-code of Gauss-Newton algorithm.

Algorithm 2.4 (*Gauss-Newton (GN) algorithm*)

Given $r(x)$;
Initialization.
$k = 0$; and
$x_k = 0$; ***For*** $k \leftarrow 0, 1, 2, \ldots$
Compute $r(x_k)$, $J(x_k)$;
$\overline{H}_k \leftarrow J(x_k)^T J(x_k)$;
$g_k \leftarrow -J(x_k)^T r(x_k)$;
Solve $\overline{H}_k p_k^{GN} = g_k$;
Compute α_k; (non exact line search)
$x_{k+1} \leftarrow x_k + \alpha_k p_k$
end(For)

The linear system (2.50) can be solved by direct methods within Gauss elimination method or LU method but for very large systems, iterative methods such as CG method may be preferable. Iterative methods have some advantages: first, they allow the storage of matrices to be avoided (only refer to some matrix-vector products). Second, they allow the algorithm to be stopped at a given precision. In any case, the linear system needs to be preconditioned

for speed, stability and precision. Some preconditioning techniques applied to medical image registration are the topic of the chapter 4 while we present first the test environment in the following chapter, Chapter 3, to prepare numerical experiments.

Chapter 3

Images and test environment

In Chapter 1 we have presented the Medical Image Registration Problem. We have highlighted some common issues and have presented some algorithms. In chapter 2 we have presented some optimization techniques that are used to tackle the MIRP issues. We have noticed that, one of the main steps which is time and memory consuming in MIRP, is the linear system resolution. Thus, we address the issue of efficient system solvers in Chapter 4. In this chapter, we present the images that we use to compare MIRP algorithms and system solvers. We present also the used tool for comparison, the performance profile, and the computer caracteristiques. Solvers are compared in Chapter 4 while the algorithms are compared later in Chapter 6.

In the Section 3.1 we present the images that constitute our test problems (the database) for simulations. In Section 3.2 we present the performance profiles tool used for benchmarking algorithms and solvers.

3.1 The images

We have 11 couples of images for numerical experiments. Each image is partitioned in a number of levels, where each level is considered as a particular test problem. Four couples of images were taken from the FAIR package and seven couples of images were provided by Hubert Meurisse, a physicist from the nuclear medicine department of the CHU-UCL-Namur Hospital of Mont Godine. These images are poorly visualized in Annexe A

Image level

A 3D image is a three dimensional array, where the size is often given by three positive integers. In most of cases, these integers express the number

of pixels, in each physical direction related to the spatial resolution of the image. The more this spatial resolution is higher, the more the image is clear but the more the treatment deals with large number of information. Thus, by assuming the intensities in the array were captured on finer grids, one can stil have a clear image by fusing adjacent cells of the image grid while averaging their intensities, more regular measurements are obtained and the size of the problem is halved. The resolution of the image is also halved.

The convention here is to assume that the level of an image is the maximal power of two in the expression of its number of pixels. Hence, for example an image is of level 1 if it is of size [2, 2, 2]. The images of sizes [128, 64, 128], [64, 64, 128] and [128, 128, 128], are of level 7.

Database images

Table 3.1 presents images used for our numerical experiments. In column 1, *num* indicates the number of the image, here alphabetic order is used. The *name* of the image in column 2 indicates, in general, the region of the body under study and sometimes the place of acquisition when the region is repeated. The *type* in column 3, indicates the acquisition modality of the image. The *format* in column 4, indicates the file format and the numeric class for storage, in column 5 the size of the image, *size*, indicates the resolution of the image. The product of the size expresses the number of voxels in the image. The system size, *syst-size*, in column 6, indicates an approximation of the size of the system induced from the image registration at the fixed level (see Section 1.4.1). The *levels* in column 7 indicate the number of levels used to register the images in a multilevel approach.

num	image	type	format	size	syst-size	levels
1	Brain	MRI	uint8	$(128, 64, 128)$	$\approx 310^7$	4
2	Chest	CT	Dicom uint8	$(512, 512, 1024)$	$\approx 810^8$	6
3	Crane	MRI	Dicom uint8	$(512, 512, 256)$	$\approx 210^7$	6
8	Foetus	CT	Dicom uint8	$(512, 512, 128)$	$\approx 210^8$	5
4	Knee	MRI	Dicom uint8	$(128, 64, 128)$	$\approx 310^6$	4
5	Lung	CT	Dicom uint8	$(512, 512, 1024)$	$\approx 810^8$	6
6	Neurocranium	CT-scan	Dicom uint8	$(256, 256, 64)$	$\approx 310^7$	5
7	PET-CT1	CT	Dicom uint8	$(256, 256, 64)$	$\approx 310^7$	5
9	PET-CT2	CT	Dicom uint8	$(512, 512, 1024)$	$\approx 810^8$	6
10	Phantom	CT	Dicom uint8	$(128, 64, 128)$	$\approx 310^6$	4
11	Mice	PET	Dicom uint8	$(128, 128, 32)$	$\approx 310^6$	3

Table 3.1: List of images that constitute our test database (given in couple).

3.2 Performance profiles

The performance profile is a tool that has gained interest of researchers by allowing to analyze and benchmark optimization solvers. Below we present an overview of this tool that will be used in Chapter 4 and in Chapter 6.

3.2.1 Overviews

Following Dolan and Moré [47], benchmarking optimization algorithms necessitates three components, (see also [48, 49]). First, we define the set of benchmark problems denoted by $\mathcal{P}$. The number of problems in this set will be denoted by n_p. In this work, this is formed by our images and we consider that for a given image, different levels constitute different problems. Thus, the number of problems is the sum of the numbers in the 7^{th} column of Table 3.1:

$$n_p = \#\mathcal{P} = 54.$$

Second, the set of optimization solvers is denoted by $\mathcal{S}$. These are designed to solve any of our problems. Third, the convergence test is a criteria that has to be verified by each algorithm (or solver) on a given problem. When

this criteria is verified, the problem is considered as solved. Although for linear system solvers, the convergence test is expected to be verified in a neighborhood of a local minimizer, this is not the case for registration algorithms. For this last, a simple threshold may be made on the functional decrease to estimate that the problem is solved. We denote by μ_f the maximal number of function evaluations [1] for each problem. A solver $s \in \mathcal{S}$ is considered as having failed to solve a problem $p \in \mathcal{P}$, if it did not verify the convergence test within μ_f iterations.

The benchmark results are produced by running each linear system solver or each registration algorithm $s \in \mathcal{S}$ on any problem $p \in \mathcal{P}$ and recording the function values in a three dimensional array

$$A(\mu_f, n_p, n_s),$$

where

$$A(f_k, p, s),\ 1 \leq k \leq \mu_f,\ 1 \leq p \leq n_p,\ 1 \leq s \leq n_s$$

contains the function value reached by the solver s on problem p at iteration k.

For any pair $(p, s) \in \mathcal{P} \times \mathcal{S}$, a performance measure $t_{p,s} > 0$ is the measured effort for solver s to verify the convergence test when solving problem p. This can be the number of function evaluations required to satisfy the convergence test, the time or other criteria. The more $t_{p,s}$ is large, the worse is the performance. The performance ratio is defined by

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s},\ s \in \mathcal{S}\}}, \tag{3.1}$$

and compares the performance of a given solver s on a problem p with respect to the best performance of any solver on this specific problem [47]. With this definition, $r_{p,s}$ is always greater than 1 and the performance ratio of the best solver s on a particular problem p is $r_{p,s} = 1$. By convention, a parameter r_M, such that, for all p and any s, $r_{p,s} \leq r_M$ is used when solver s fails to satisfy the convergence test. The parameter r_M can be considered ∞.

Let us consider $\alpha \in \mathbb{R}$ and define [47]

$$\rho_s(\alpha) = \frac{1}{n_p} size\{p \in \mathcal{P} : r_{p,s} \leq \alpha\} \tag{3.2}$$

as the probability for solver $s \in \mathcal{S}$ that a performance ratio $r_{p,s}$ is within a factor α of the best possible ratio. The function $\rho_s(\alpha)$ is then a (cumulative) distribution function for performance ratio and informs how well the solver s performs relative to the other solvers in $\mathcal{S}$ on the set of problems $\mathcal{P}$. In general, $\rho_s(\alpha)$ indicates the proportion of problems with a performance ratio at most α.

1. Here function evaluation denotes both the evaluation of the residual norm for linear system solvers and the functional evaluation for registration algorithms. Similarly, the term solver may designates both a linear system solver and a registration algorithm.

In particular, $\rho_s(1)$ denotes the fraction of problems for which solver s performs the best and $\rho_s(\alpha)$, for α sufficiently large, is the fraction of problems solved by the solver s. The preferable solvers are, thus, those with high values for $\rho_s(\alpha)$.

Consider that $t_{p,s} > 0$ measures the effort for solver s to verify the convergence test when solving problem p while the best solver $\hat{s}$ requires $\hat{t}_{p,\hat{s}}$ for the same problem. The factor α in $\rho_s(\alpha)$ is such that

$$t_{p,s} \geq \alpha \hat{t}_{p,\hat{s}}.$$

This means, the solver s requires at least α times the efforts of the best solver to verify the convergence test.

Solvers are compared in Chapter 4 while the algorithms are compared later in Chapter 6. We present in Chapter 4 and in Chapter 6 some numerical experiments to illustrate the behaviour of algorithms and solvers described in chapters 4 and in Chapter 5.

3.3 specific toolboxes

For this work, we have used many matlab classical toolboxes. In addition to the FAIR package, we have used the following specific toolboxes. *Tensorlab toolbox* (see [50]) a toolbox for tensor computations and signal processing that we used to a full array in a sparse one.

`htucker toolbox` (see [51]) for linear systems in tensor formats.

`tt-toolbox` (see [52]) for linear system in Tensor-Train formats.

3.4 Characteristics of the computer

The tests in this thesis are implemented in *Matlab* and we have two test environments. The first tests, concerned by relatively small and medium images, were performed on a local computer with one processor intel Core$^{TM}i5$, $3.2GHz \times 4$.

The second test environment, concerned by larger images, were the hercules machine, one of the CECI clusters [2] The allocated resources in this environment are described in Table 3.2

4 processors	CPU = 4
Memory	16GB per processor
Compiler	*Matlab* on slurm smp

Table 3.2: Host computer characteristic

2. `http://www.ceci-hpc.be/hercules` and `http://www.ptci.unamur.be/hercules` .

Chapter 4

Linear system solvers and preconditioning for 3D MIRP

As stated in the previous chapters, most nonrigid registration processes require resolution of large linear systems. For nonparametric registration techniques, these linear systems are derived from physical principles, modelled as Partial Differential Equations (PDEs), whose resolution allows to obtain a smooth displacement field. On the one hand, such linear systems are generally ill-conditioned. This leads to low convergence rates of the algorithms used to solve the problem, or to inaccurate solutions. On the other hand, many applications of registration algorithms such as cancer screening in intraoperative brain shift estimation (in image guided neurosurgery) require faster registration algorithms [53]. Thus, there is a need to provide efficient system solvers especially for deformable nonparametric registration applied to high-resolution $3D$ images. In such applications, the system resolution is indeed among the most expensive steps.

One way of designing efficient system solvers is the analysis of structures of general operators or matrices used by the algorithms. Getting insights into those structures may enable a reduction of the number of expensive operations such as matrix-vector products and thus to speed up the registration process. In this chapter, we are especially interested in speeding up the Conjugate Gradient (CG) algorithm by providing well suited preconditioners (see Section 2.2).

For designing efficient system solvers, one may analyze structures of general operators or matrices used by the algorithms. In our case, although these systems are often sparse and structured, they are very large and ill-conditioned. Thus, their solvers are time consuming and their complexity in operation counts is polynomial. As a consequence, fast and superfast direct methods reveal numerical instabilities and thus lead to breakdowns or inaccurate solutions. The most used alternative are iterative system solvers that enable a reduction

of the number of expensive operations such as matrix-vector products and thus a speed up of the registration process. Although iterative solvers provide only an approximation of the solution, they are well suited for very large systems when cheap and well suited preconditioners are available. Preconditioners may be stationary (Jacobi, Gauss-Seidel or SOR) and non-stationary (polynonmial or low-rank tensors).

In this chapter, we present roughly the general techniques of preconditioning linear systems of the form

$$Ax = b, \tag{4.1}$$

where $A \in \mathbb{R}^{N\times N}$ is large, sparse and Symmetric Positive-Definite (SPD) constructed from MIRP. This matrix approximates, in some way, the Hessian of the functional (1.65). Except in Section 4.5 where the system is derived from a Gauss-Newton strategy, in other sections the approximation is based only on the differential operator (see (1.58) and (1.58)). The vectors x, b belong to $\mathbb{R}^N$ with b that contain information about the images to be registered (see 1.59).

4.1 Systems from Elastic and Diffusion models

Let us suppose that the registration problem consists of deforming an elastic material with linear elasticity [24, 32]. From the formulation (1.65), we need a displacement field, $u = [u^{(1)}, u^{(2)}, \ldots, u^{(d)}]$ [1], that minimizes the elastic linear energy, expressed by

$$E_{_R}(u) = \int_\Omega \frac{\mu}{4} \sum_{j,k=1}^{d} \left(\partial_{x_j} u^{(k)} + \partial_{x_k} u^{(j)}\right)^2 + \frac{\lambda}{2}(divu)^2 dx, \tag{4.2}$$

where the scalars μ and λ are the so-called Lamé constants [24]. These scalars refer to the rigidity of the elastic body under deformations. The deformation of an elastic body assumes an external force has been applied on it, while the strain of the body, related to its inner stress, is opposed to this external force. The final shape of the body will result from an equilibrium of the external forces and the inner stress.

For physical interpretation, we refer to [24, pp. 84-94]. Note that, this linear elastic energy (4.2 is invariant with respect to rigid transformations. This enables the use of a rigid pre-registration before elastic deformation. In addition, with homogenous Dirichlet or Neumann boundary conditions, this functional is positive definite [24], that is, $\forall u \neq 0, E_{_R}(u) > 0$, and symmetric by construction. This leads to symmetric positive definite matrices well suited for many linear system solvers. The particular case of equation (4.2) when

1. The notation $u^{(k)}$ indicates the component of u in the k^{th} direction, $k = 1, 2, \ldots, d$.

$d = 3$ writes:

$$E_{R}(u) = \int_{\mathbb{R}^3} \frac{\lambda}{2} \left(\partial_{x_1} u^{(1)} + \partial_{x_2} u^{(2)} + \partial_{x_3} u^{(3)}\right)^2 + \mu \left\{ \left(\partial_{x_1} u^{(1)}\right)^2 + \left(\partial_{x_2} u^{(2)}\right)^2 + \left(\partial_{x_3} u^{(3)}\right)^2 \right\} + \frac{\mu}{2} \left\{ \left(\partial_{x_1} u^{(2)} + \partial_{x_2} u^{(1)}\right)^2 + \left(\partial_{x_1} u^{(3)} + \partial_{x_3} u^{(1)}\right)^2 + \left(\partial_{x_2} u^{(3)} + \partial_{x_3} u^{(2)}\right)^2 \right\} dx.$$

In this general approach, another advantage is that regularisation by linear elastic energy minimization takes into account the coupling between coordinates during deformations. This means that it considers the fact that a deformation along one axis impacts positions of points with respect to the other axes.

The Euler-Lagrange equation is derived from the fact that, (see for e.g, [24, 34]), at a local minimizer function of a differentiable functional, the derivative of this functional is zero. Since we are interested in finding a local minimizer of the functional (1.65), we may determine a small and smooth displacement v where the Gâteaux derivative of the functional is zero.

The following theorem states the Gâteaux derivative of the registration cost functional with elastic regularisation. This allows the deformation that minimizes the linear elastic energy to be computed while fitting images to be registered. The theorem uses the Euclidean scalar product $\langle ., . \rangle_{\mathbb{R}^d}$ on $\mathbb{R}^d$ and the L^2-norm $\|.\|_{L^2}$, that is the Euclidean norm for square-integrable functions. The notation $(C^2(\mathbb{R}^d))^d$ expresses the set of functions whose components are at least twice continuously differentiable each on $\mathbb{R}^d$.

Theorem 4.1 (*[24]*)

Let

$$J[u] = \frac{1}{2}\|I_m[x - u(x)] - I_f(x)\|_{L^2}^2 + \frac{\lambda}{2} E_{R}(u),$$

be a functional where $E_{R}(u)$ is given by (4.2). Given a displacement field $u \in (C^2(\mathbb{R}^d))^d$ and a perturbation $v \in (C^2(\mathbb{R}^d))^d$, the Gâteau derivatives of J is given by

$$dJ[u, v] = \int_{\Omega} \langle F - \mu \Delta u - (\mu + \lambda) \nabla div u, v \rangle_{\mathbb{R}^d} dx. \tag{4.3}$$

where $F = F(x_1, x_2, x_3)$ is given by (1.59).

Proof. See [24, p. 100] □

The theorem above can be used to derive the Euler-Lagrange equation of the process that describes the equilibrium between the external forces and the inner strain of the body. In an optimization context, we are interested in the stationary point of $J[u]$ that may be sufficient as minimizer of (1.65). Thus, we need to find a displacement field u such that $dJ[u, v] = 0$. That is, we need to solve the equation

$$\int_{\Omega} \langle F - \mu \Delta u - (\mu + \lambda) \nabla div u, v \rangle_{\mathbb{R}^d} dx = 0. \tag{4.4}$$

Since this equality has to be verified by any perturbation v that is not necessarily zero, the first quantity of the inner product above must vanish. We have

$$(\mu\Delta + (\mu + \lambda)\nabla div)u = F. \tag{4.5}$$

This allows us to define the elastic operator as

$$A_{el} = \mu\Delta + (\mu + \lambda)\nabla div. \tag{4.6}$$

To determine u that verifies (4.4), we have to solve the linear system (4.5) that can be rewritten

$$A_{el}u = F. \tag{4.7}$$

We are interested in providing efficient numerical resolution of equation (4.7). To this equation, we have to add a second equation obtained by the diffusion model that we discussed roughly in section (1.3.8). To establish the equation from the diffusion model, we refer to [24, 4, 10]. Here, we decide to derive the equation (forgetting physical interpretation) by considering $\mu = 1$ and $\lambda = -1$ in the elastic model (4.7). The operator of the equation in (4.7) becomes

$$A_{dif} = \Delta, \tag{4.8}$$

where Δ is the 3-dimensional Laplace operator. Then we need to efficiently solve a multidimensional Laplacian linear equation

$$A_{dif}u = F. \tag{4.9}$$

After we have formed the linear systems (4.7) and (4.9), it remains to design an appropriate solver for their solutions. Note that these systems are, in general, very large. For example, the registration of two 3D images of size $[512, 512, 512]$ leads to a linear system with coefficient matrix $A \in \mathbb{R}^{3N\times 3N}$ where $N = 2^{27}$. This means, we have to solve a linear system that is sufficiently large with, roughly, 410^8 unknowns . It is obvious in this case that, the convergence rate of a registration algorithm is related to the efficiency of its linear system solver.

In linear algebra, large scale linear system solvers seek for algorithms which should require only linear complexity (in term of operations count), especially for matrix-vector product, and if possible, the matrix inversion. Since this may be too difficult for general matrices, one may work with special families of matrices. An ideal family of such matrices is the family of diagonal matrices [54].

Observe that, for matrices A, $B \in \mathbb{R}^{m\times m}$, $m \in \mathbb{N}$ and a vector $x \in \mathbb{R}^m$, only diagonal matrices allow the standard operations $Ax, A+B, A*B$ and A^{-1}, to be performed with linear complexity (in term of operations count) [54].

In addition to diagonal matrices, sparse matrices form another interesting family for algorithms that require only matrix-vector product as "expensive" operation.

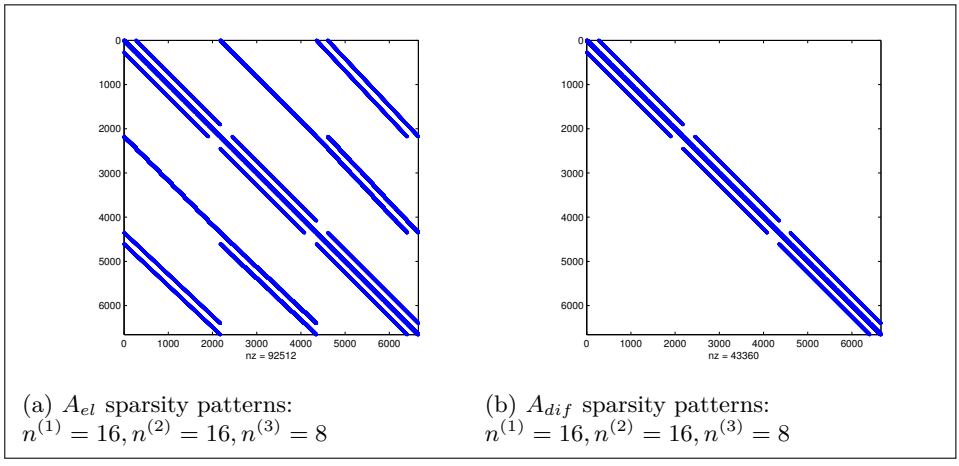
Thus, one way of designing an efficient linear system solver is to exploit the structure of the matrix system. For specific structures, such as diagonal, block diagonal, sparse, Toeplitz or Hankel matrices, there exist direct fast and very fast solvers (see e.g, [49]).

It is known that direct solvers can prove to be robust and attractive for their predictability. For these methods the requirements in term of computing time and storage can be known in advance [55]. Thus, to solve the large linear systems (4.7) and (4.9), one may have the temptation of using direct methods. Since the efficiency of direct methods relie on specific structure of the system matrix, it is of importance to analyse the structure of matrices at hand. Below, we present the structure of our system matrices A_{el} and A_{dif}.

4.1.1 Structure and sparsity of A_{el} and A_{dif}

Since A_{el} and A_{dif} are derived from discretization of PDEs, they are naturally sparse. The sparsity patterns may change slightly with respect to the discretization method and boundary conditions, but in general, these matrices stay sparse.

Figure 4.1 displays the sparsity patterns of A_{el} (left) and A_{dif} (right) for a three-dimensional finite differences discretization. The Dirichlet boundary conditions are assumed for both systems. The size of the matrices is $[6144 \times 6144]$ but nonzeros elements are reduced, respectively, to 92512 and 43360 that are small compared to $(6144)^2$ that should have dense matrices.



(a) A_{el} sparsity patterns: $n^{(1)} = 16, n^{(2)} = 16, n^{(3)} = 8$

(b) A_{dif} sparsity patterns: $n^{(1)} = 16, n^{(2)} = 16, n^{(3)} = 8$

Figure 4.1: Sparsity of elastic and diffusion operators

Now, it is important to know where are located the non zero elements. One can verify that, by developing the relation (4.2), the linear operator A_{el} in $3D$

can be written

$$\begin{aligned} A_{el} &= \begin{pmatrix} (\lambda+2\mu)\partial_{x_1x_1}u^{(1)} + \mu\partial_{x_2x_2}u^{(1)} + \mu\partial_{x_3x_3}u^{(1)} \\ \mu\partial_{x_1x_1}u^{(2)} + (\lambda+2\mu)\partial_{x_2x_2}u^{(2)} + \mu\partial_{x_3x_3}u^{(2)} \\ \mu\partial_{x_1x_1}u^{(3)} + \mu\partial_{x_2x_2}u^{(3)} + (\lambda+2\mu)\partial_{x_3x_3}u^{(3)} \end{pmatrix} \\ &+ (\lambda+\mu)\begin{pmatrix} \partial_{x_1x_2}u^{(2)} + \partial_{x_1x_3}u^{(3)} \\ \partial_{x_1x_2}u^{(1)} + \partial_{x_2x_3}u^{(3)} \\ \partial_{x_1x_3}u^{(1)} + \partial_{x_2x_3}u^{(2)} \end{pmatrix} \end{aligned} .$$

Observe that, with appropriate discretizations based on finite differences methods, A_{el} is a real matrix of size $3N\times 3N$ where $N = n^{(1)}n^{(2)}n^{(3)}$ and $[n^{(1)}, n^{(2)}, n^{(3)}]$ expresses the size of the image. For more on these discretization schemes, we refer to J. Modersitzki [10].

Then, using finite differences methods, let us set

$$\begin{aligned} A^{(11)} &= (\lambda+2\mu)\partial_{x_1x_1}u^{(1)} + \mu\partial_{x_2x_2}u^{(1)} + \mu\partial_{x_3x_3}u^{(1)} \\ A^{(12)} &= (\lambda+\mu)\partial_{x_1x_2}u^{(1)} \\ A^{(13)} &= (\lambda+\mu)\partial_{x_1x_3}u^{(1)} \\ A^{(21)} &= (\lambda+\mu)\partial_{x_2x_1}u^{(2)} \\ A^{(22)} &= \mu\partial_{x_1x_1}u^{(2)} + (\lambda+2\mu)\partial_{x_2x_2}u^{(2)} + \mu\partial_{x_3x_3}u^{(2)} \\ A^{(23)} &= (\lambda+\mu)\partial_{x_2x_3}u^{(2)} \\ A^{(31)} &= (\lambda+\mu)\partial_{x_3x_1}u^{(3)} \\ A^{(32)} &= (\lambda+\mu)\partial_{x_3x_2}u^{(3)} \\ A^{(33)} &= \mu\partial_{x_1x_1}u^{(3)} + \mu\partial_{x_2x_2}u^{(3)} + (\lambda+2\mu)\partial_{x_3x_3}u^{(3)} \end{aligned} .$$

Then, we can approximate these second derivatives by

$$\partial_{x_jx_j}u(x) = \frac{u(x+h_je_j) - 2u(x) + u(x-h_je_j)}{h_j^2} + \mathcal{O}(h_j^2) \tag{4.10}$$

and

$$\begin{aligned} \partial_{x_jx_k}u(x) = \frac{1}{4h_jh_k}u(x+h_je_j+h_ke_k) - u(x-h_je_j+h_ke_k) \\ -u(x+h_je_j-h_ke_k) + u(x-h_je_j-h_ke_k) + \mathcal{O}(h_j^2+h_k^2). \end{aligned} \tag{4.11}$$

To compute efficiently these quantities, one can use 3-by-3-by-3 stencil matrices by defining

$$\alpha_{j,k,l} = \begin{cases} -2(\lambda+4\mu), & \text{if} \quad j=k=l=2, \\ (\lambda+2\mu), & \text{if} \quad j=1,3,\ k=l=2, \\ \mu, & \text{if} \quad j=k=2,\ l=1,3, \\ \mu, & \text{if} \quad j=l=2,\ k=1,3, \\ 0, & \text{other wise} \end{cases}$$

and setting

$$\beta_{j,k,l} = \frac{\lambda+\mu}{4}\begin{cases} 1, & \text{if} \quad j=2 \text{ and } (k=l=1 \text{ or } k=l=3), \\ -1, & \text{if} \quad j=2 \text{ and } (k=1,\ l=3 \text{ or } k=3,\ l=1), \\ 0, & \text{other wise.} \end{cases}$$

$S^{(11)}_{j,k,l} = \alpha_{j,k,l}$	$S^{(12)}_{j,k,l} = \beta_{l,j,k}$	$S^{(13)}_{j,k,l} = \beta_{k,l,j}$
$S^{(21)}_{j,k,l} = \beta_{l,j,k}$	$S^{(22)}_{j,k,l} = \alpha_{k,l,j}$	$S^{(23)}_{j,k,l} = \beta_{j,k,l}$
$S^{(31)}_{j,k,l} = \beta_{k,l,j}$	$S^{(32)}_{j,k,l} = \beta_{j,k,l}$	$S^{(33)}_{j,k,l} = \alpha_{l,j,k}$

Table 4.1: $3D$ stencil arrays.

Based on these definitions, we can define three dimensional arrays, as stencil arrays. This is presented in Table 4.1.

Observe that the action of A_{el} on the displacement field u can be seen as a convolution of each component of u by the corresponding stencil matrix, that is,

$$A_{el}[u(x)] = \begin{pmatrix} S^{(11)} * u^{(1)}(x) & S^{(12)} * u^{(2)}(x) & S^{(13)} * u^{(3)}(x) \\ S^{(21)} * u^{(1)}(x) & S^{(22)} * u^{(2)}(x) & S^{(23)} * u^{(3)}(x) \\ S^{(31)} * u^{(1)}(x) & S^{(32)} * u^{(2)}(x) & S^{(33)} * u^{(3)}(x) \end{pmatrix},$$

where $*$ is the convolution operation. For instance,

$$S^{(11)} * u^{(1)}_{j,k,l} = \sum_{r,s,t=-1}^{1} S^{(11)}_{2+r,2+s,2+t} * u^{(1)}_{j+r,k+s,l+t}.$$

This can also be seen as a product of a block matrix by the vectorized function u, in the form

$$A_{el}[u(x)] = \begin{pmatrix} A^{(11)} & A^{(12)} & A^{(13)} \\ A^{(21)} & A^{(22)} & A^{(23)} \\ A^{(31)} & A^{(32)} & A^{(33)} \end{pmatrix} \begin{pmatrix} u^{(1)}(x) \\ u^{(2)}(x) \\ u^{(3)}(x) \end{pmatrix}$$

The elastic matrix is then a block-matrix

$$A_{el} = \begin{pmatrix} A^{(11)} & A^{(12)} & A^{(13)} \\ A^{(21)} & A^{(22)} & A^{(23)} \\ A^{(31)} & A^{(32)} & A^{(33)} \end{pmatrix}, \tag{4.12}$$

while the diffusion operator A_{dif} is the three dimensional Laplacian. They are both naturally symmetric positive definite.

While it is obvious that the matrices A_{el} and A_{dif} are not diagonal, they are clearly highly structured and sparse. Thus, cheap transformations may allow to diagonalize them and then work with diagonal or block diagonal matrices. The most known cheap transformations are Fast Fourier Transforms (FFTs) or their related Discrete Cosine Transforms (DCTs). With such fast methods, it is possible to solve the linear systems (4.7) and (4.9) with complexity $\mathcal{O}(Nlog^2N)$ [56] or even $\mathcal{O}(NlogN)$ [24] where $N \in \mathbb{N}$ is the number of voxels of the image. Below, we present DCT of the operator (4.8) and FFT of the operator (4.6).

4.1.2 Factorization of A_{el} and A_{dif}

Factorisation of A_{el}

To decompose A_{el} we follow the procedure in [24], the block-matrix formed by $A^{(jk)},\ j,k = 1,2,\ldots,d$ can be transformed using circulant matrices defined below. The advantage of using circulant matrices is that they are diagonalizable by the Fourier matrix and allow equivalence between the convolution operation and the matrix-vector product. Thus, we need to use circulant matrices in order to transform A_{el} in a block diagonal matrix. This means a block-matrix whose blocks are all diagonals. The definition of a circulant matrix is stated below:

Definition 4.2 (*circulant matrix*)

Given $m \in \mathbb{N}$, a matrix is called circulant, if it is generated by an m-vector $c = [c_0, c_1, \ldots, c_{m-1}]$ such that

$$C_m = \begin{pmatrix} c_0 & c_1 & \ldots & c_{m-2} & c_{m-1} \\ c_{m-1} & c_0 & & & c_{m-2} \\ \vdots & \ddots & \ddots & \ddots & \\ c_2 & & & c_0 & c_1 \\ c_1 & c_2 & \ldots & c_{m-1} & c_0 \end{pmatrix}, \tag{4.13}$$

where it can be seen that each row (respectively each column) is a cyclic shift of the preceding row (respectively column).

The specific circulant matrix we will use here is

$$C_m = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & \ddots & \ddots & \\ 0 & & & & 1 \\ 1 & 0 & \ldots & 0 & 0 \end{pmatrix}. \tag{4.14}$$

It can be shown that any circulant matrix C_m is unitary. This means that $C_m^T C_m = C_m C_m^T = I_m$ and $C_m^{-1} = C_m^T$. In addition, it is known that any circulant matrix can be diagonalized by the Fourier matrix

$$F_m = \frac{1}{\sqrt{m}} \left(w_m^{(j-1)(k-1)} \right)_{j,k=0}^{m-1}, \tag{4.15}$$

where $w_m = e^{-2\pi i/m}$ is the m^{th} root of unity and $i^2 = -1$. This is stated in the following result demonstrated in [57].

Proposition 4.3

Let $m \in \mathbb{N}$, F_m be the Fourier matrix defined in (4.15) *and*

$$\Lambda_m = diag(w_m^0, w_m^{(1)}, \ldots, w_m^{m-1}),$$

be a diagonal matrix. Then,

1. *the Fourier matrix is unitary, that is,* $F_m^{-1} = F_m^H$ *where* F_m^H *indicates the conjugate transpose of the matrix* F_m,
2. *the circulant matrix* C_m *is diagonalized by* F_m, *that is*

$$F_m^H C_m F_m = \Lambda_m, \tag{4.16}$$

3. *any linear combination of circulant matrices* $Z_m = \sum_{j=1}^{m} \alpha_j C_m^j$ *is diagonalized by* F_m, *with*

$$F_m^H Z_m F_m = \sum_{j=1}^{m} \alpha_j (\Lambda_m)^j. \tag{4.17}$$

Using this proposition, it is demonstrated in [24] that the blocks of the matrix A_{el} in (4.12) can be written as block-circulant, that is, with blocks that are circulants and that those blocks can be factorized using Fourier matrices.

This means that, the Fourier matrix allows to transform the initial matrix into a block matrix whose blocks are diagonal.

Finally the matrix $A \in \mathbb{R}^{(3n^{(1)}n^{(2)}n^{(3)})\times(3n^{(1)}n^{(2)}n^{(3)})}$ (cfr (4.12)) allow the transformation

$$F^H A F = \begin{pmatrix} D^{(11)} & D^{(12)} & D^{(13)} \\ D^{(21)} & D^{(22)} & D^{(23)} \\ D^{(31)} & D^{(32)} & D^{(33)} \end{pmatrix}$$

where $F = I_3 \otimes F_{n^{(3)}} \otimes F_{n^{(2)}} \otimes F_{n^{(1)}}$, and $D^{(pq)}, p, q \in \{1, 2, 3\}$ are block diagonal matrices.

Now this allows the linear system (4.7) to be solved by applying explicit fast inversion of block diagonal matrices $D^{(pq)}$, $p, q = 1, 2, 3$ and applying some FFTs [24].

However, the matrix A or matrices $D^{(pq)}$ to be inverted by the fast or superfast algorithms might be singular or very ill-conditioned [24, p. 109]. This situation often leads to break-down or causes numerical instability that may lead to inaccurate solutions. In the case of singularity, J. Modersitzki [24] proposed to compute the Moore-Penrose pseudo inverses to overcome the problem. He proposed the algorithm below, where for direct use in matlab, the command $fft3$ reffers to the Fast Fourier Transforms in three dimensional space.

Algorithm 4.1 (*[24]*)

Given $x = [x^{(1)}, x^{(2)}, x^{(3)}]$, $F = [f_1(x), f_2(x), f_3(x)]$ *we are looking for* $u = [u^{(1)}, u^{(2)}, u^{(3)}]$ *such that* $A_{el}u(x) = F(x)$

For $r = 1 : 3$

$\widetilde{F}^{(r)} = fft3(f_r(x))$;

end

For $j = 1 : n^{(1)}$, *for* $k = 1 : n^{(2)}$, *for* $l = 1; n^{(3)}$

$$\textit{Solve} \begin{pmatrix} \widetilde{u}^{(1)}_{l,k,j} \\ \widetilde{u}^{(2)}_{l,k,j} \\ \widetilde{u}^{(3)}_{l,k,j} \end{pmatrix} = \begin{pmatrix} D^{(11)}_{l,k,j} & D^{(12)}_{l,k,j} & D^{(13)}_{l,k,j} \\ D^{(21)}_{l,k,j} & D^{(22)}_{l,k,j} & D^{(23)}_{l,k,j} \\ D^{(31)}_{l,k,j} & D^{(32)}_{l,k,j} & D^{(33)}_{l,k,j} \end{pmatrix}^{\dagger} \begin{pmatrix} \widetilde{F}^{(1)}_{l,k,j} \\ \widetilde{F}^{(2)}_{l,k,j} \\ \widetilde{F}^{(3)}_{l,k,j} \end{pmatrix}$$

end

For $r = 1 : 3$

$$u^{(r)} = fft3^{-1}(\widetilde{u}^{(r)});$$

end

where $D^{\dagger}$ indicates the Moore-Penrose pseudo-inverse of the matrix D. This algorithm solves the linear system with numerical complexity $\mathcal{O}(NlogN)$, $N = \prod_{k=1}^{3} n^k$.

Factorisation of A_{dif}

Observe that for the diffusion model (4.8) the transformation may be simplified because the mixte derivatives are zeros. Then, we can use variables separability and thus the Kronecker product defined in (1.21). With this configuration, A_{dif} is a Laplacian operator in $3D$ that writes

$$A_{dif} = I_{n^{(3)}} \otimes I_{n^{(2)}} \otimes \Delta_{n^{(1)}} + I_{n^{(3)}} \otimes \Delta_{n^{(2)}} \otimes I_{n^{(1)}} + \Delta_{n^{(3)}} \otimes I_{n^{(2)}} \otimes I_{n^{(1)}}, \quad (4.18)$$

where $\Delta_{n^{(k)}}$, $n^{(k)} \in \mathbb{N}$, $k = 1, 2, 3$, is the unidimensional Laplacian operator given by

$$\Delta_{n^{(k)}} = \frac{1}{{h^{(k)}}^2} \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{n^{(k)} \times n^{(k)}}.$$

The variable separability that allows the expression (4.18) enables to study and compute eigenelements of the operator A_{dif} via unidimensional Laplacian. Thus, the matrix A_{dif} can be diagonalized by discrete sine or cosine transforms. In what follows, we use the Discrete Sine Transfor (DST). The use of this transformation is twofold. At the one hand, they provide diagonal matrices for efficient direct linear system solvers. At the other hand, they provide analytic expressions of eigenvalues and associated eigenvectors that can be used for designing efficient preconditioners for iteratives solvers (these are presented in the next section).

Let us define

$$\theta_{j_k} = \frac{j_k \pi}{(n^{(k)} + 1)}, \quad j_k = 1 : n^{(k)}, , \quad k = 1, 2, 3. \quad (4.19)$$

It can be shown that, the eigenvalues of A_{dif} are given by

$$\lambda_{j_1,j_2,j_3} = 4\left[sin^2\left(\frac{\theta_{j_1}}{2}\right) + sin^2\left(\frac{\theta_{j_2}}{2}\right) + sin^2\left(\frac{\theta_{j_3}}{2}\right)\right], \tag{4.20}$$

for $j_k = 1, 2, \ldots, n^{(k)}$, $k = 1, 2, 3$. Equivalently, by using

$$2sin^2\left(\frac{\theta}{2}\right) = 1 - cos(\theta),$$

$$\lambda_{j_1,j_2,j_3} = 6 - 2\left[cos\left(\theta_{j_1}\right) + cos\left(\theta_{j_2}\right) + cos\left(\theta_{j_3}\right)\right].$$

The associated eigenvectors are Kronecker product of unidimensional eigenvectors. The unidimensional eigenvectors associated to the j_k^{th} eigenvalue of the operator $\Delta_{n^{(k)}}$ is given by

$$v_{i_k} = \sqrt{2/(n^{(k)}+1)}\sin\left(i_k\theta_{j_k}\right), j_k = 1 : n^{(k)}.$$

The eigenvector associated to the 3-dimensional Laplacian is then

$$v_{i_1,i_2,i_3} = v_{i_1} \otimes v_{i_2} \otimes v_{i_3}. \tag{4.21}$$

If

$$V_{n^{(k)}} = \left(\sqrt{2/(n^{(k)}+1)}\sin\left(i_k\theta_{j_k}\right)\right)_{i_k,\,j_k=1}^{n^{(k)}}$$

is the orthonormal matrix of the unidimensional Laplacian eigenvectors, the orthonormal matrix of the 3-dimensional Laplacian is given by

$$V = V_{n^{(3)}} \otimes V_{n^{(2)}} \otimes V_{n^{(1)}}, \tag{4.22}$$

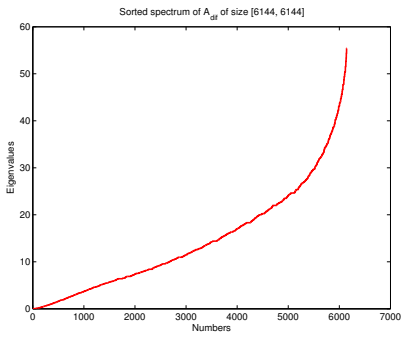
and one can verify that

$$V^TV = I_N, \quad \text{and} \quad V^TA_{dif}V = \Lambda, \tag{4.23}$$

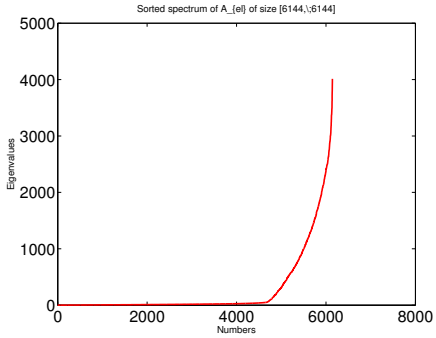
where Λ is a diagonal matrix whose elements are eigenvalues defined by (4.20) and $N = n^{(1)}n^{(2)}n^{(3)}$.

Figure 4.2 illustrates the eigenvalues distribution of both A_{dif}, Laplace operator, (top left) and A_{el} (top right) with the parameters $\mu = 1$ and $\lambda = 1$. One can see that they have almost the same distribution but the upper bound of A_{el} depends also on the parameters λ and μ from the formulation. In the figure, one can see that the eigenvalues are not uniformly distributed. Instead, there is concentration of smaller eigenvalues and concentration of relatively higher eigenvalues. The subfigures (4.2a), and (4.2b) display ordered eigenvalues of A_{dif} and A_{el} for matrices of size 6144×6144 The subfigures (4.2c), and (4.2d) make the x-axis on a log scale to show that there are many eigenvalues closer to zero for A_{dif} but smaller for A_{el}, respectively, and no upper bound for the latter. The condition number for these specific cases are $k(A_{dif}) = 10^{15}$ and $k(A_{el}) = 10^6$. Notice that, for the matrix (4.8), when $n^{(k)}$ is relatively large (e.g $n^{(k)} \geq 100$, $\forall k = 1, 2, 3$), this matrix has identical eigenvalues in practice and thus low-rank.

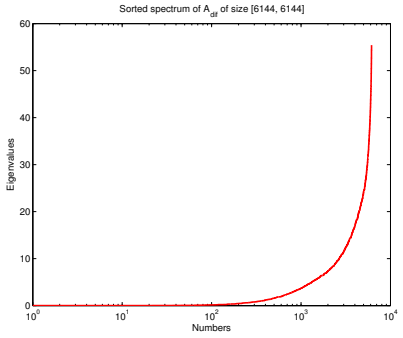
In addition, the more the size $N = n^{(1)}n^{(2)}n^{(3)}$ increases, the more there are quasi similar eigenvalues that reduce the number of clusters. On the other hand, the ratio between the smallest and the highest eigenvalues increases. This is due to the fact that, for $j_k = n^{(k)}$, $k = 1, 2, 3$, the lower bound decreases and tends to zero.



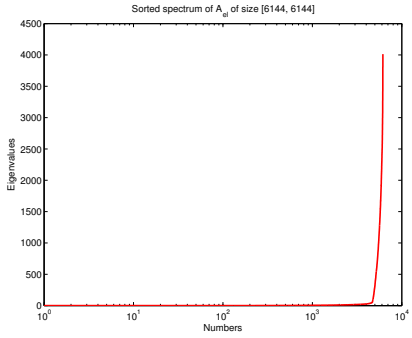
(a) Sorted spectrum of A_{dif} ($3D$ Laplace operator)



(b) Sorted spectrum of A_{el} Elastic operator



(c) Sorted spectrum of A_{dif} with log scaled x-axis.



(d) Sorted spectrum of A_{el} with log scaled x-axis.

Figure 4.2: Eigenvalues distribution of A_{dif} ((4.2a) and (4.2c)) and A_{el}((4.2b) and (4.2d)). The size of both matrices is 6144×6144.

We have factorized our matrices and direct fast methods have explored these factorisations to solve the linear systems (see for example [58]). However, it is known that the matrices A_{el} and A_{dif} or even the blocks $D^{(pq)}$ are very ill conditioned. Thus, these fast and superfast direct methods have numerical instabilities and lead to breakdowns or inaccurate solutions. An alternative is the use of iterative algorithms. But one should note that, iterative solvers provide only an approximation to the solution of the linear system and may introduce further inner iterations. Since the matrices A_{el} and A_{dif} are symmetric and positive definite, the Conjugate Gradient (CG) algorithm described in Section 2.2 is the iterative method of choice for these linear systems.

To have an iterative method that is competitive with the fast and superfast direct methods, we have to provide an algorithm with at most linear complexity. This is possible, only if there is possibility to build a "low-cost" preconditioner, in term of operation counts and that should be, in addition, easy to apply. We address certain preconditioning techniques in the section below.

4.2 Introduction to preconditioning techniques

In Section 2.2, we have presented certain results on the CG convergence rate. In those studies, the analysis of rounding errors is not taken into account although their effects may be unacceptable in practice [59]. In the following section, we present a rough analysis of the CG convergence rate in presence of rounding errors and we present certain preconditioning techniques that may address this issue.

4.2.1 Convergence of the CG algorithm in finite precision

The problem of rounding errors for the CG algorithm has been addressed by many authors in the literature (see for example [60, 61, 62]). In the aim of enphasizing on the role of the right-hand side in the convergence rate of the CG algorithm, we present in this subsection, some results on the CG behaviour in finite precision following the papers [63] and [64]. The analysis of the CG convergence rate presented here does not focus on the process of achieving the solution but on getting insight into the earlier behaviour of the algorithm.

One of the sources of rounding errors in the CG algorithm may be the loss of orthogonality of the residual vectors. The CG algorithm is known to be sensitive to this loss of orthogonality, since this changes the theoritical convergence properties of the CG algorithm. In particular, the loss of orthogonality delays the convergence and may even prevent the attainable accuracy [63]. However, it has been shown that, for some matrices, a small change in the eigenvalue distribution can lead to a large change in the sensitivity of the CG algorithm with respect to rounding errors [65]. Thus, after a brief analysis of the loss of orthogonality of the residual vectors in the CG algorithm, we will present a way of changing the coefficient matrix of the system such that the eigenvalue distribution of the new matrix becomes less sensitive to rounding errors. This is a way of preconditioning the system addressed in this chapter.

Let us assume that the CG algorithm is designed to solve the linear system (2.23). Then, we present below the convergence behaviour of the CG algorithm at iteration $k > 0,\ k \in \mathbb{N}$. As presented in Section (2.2), given an initial guess x_0, at the iteration k, the CG algorithm aims to minimize the energy norm of the error on the Krylov subspace $\mathcal{K}_k(A, r_0)$ (see 2.24).

Thus, the convergence rate of the CG algorithm can be analyzed by comparing the current error $\|x^* - x_k\|_A^2$ to the initial error $\|x^* - x_0\|_A^2$ in the A-norm. For this purpose, let us notice three equivalent expressions stated in the following remark.

Remark 4.1

Let x^* denotes the solution of the linear system $Ax = b$, that is, $Ax^* = b$. Consider $r = b - Ax$, $\|r\|_{A^{-1}}^2 = r^T A^{-1} r$ and $q(x) = \frac{1}{2} x^T Ax - b^T x$. Then, the expressions

$$\|r\|_{A^{-1}}^2, \tag{4.24}$$

$$\|x - x^*\|_A^2 \tag{4.25}$$

and

$$2q(x) + b^T A^{-1} b \tag{4.26}$$

are equivalent.

This can be proved by an easy verification with basic linear algebra operations.

The CG algorithm analyzed here is computationally based on the three-two terms recurrences

$$\begin{cases} r_k &= r_{k-1} - \alpha_k A p_{k-1}, \\ x_k &\in x_0 + \mathcal{K}_k(A, r_0), \\ p_k &= r_k + \beta_k p_{k-1}, \end{cases} \tag{4.27}$$

With the CG properties stated in Theorem 2.9. The residual vectors $\{r_0, r_1, \ldots, r_{k-1}\}$ form an orthogonal basis and search direction vectors $\{p_0, p_1, \ldots, p_{k-1}\}$ form an A-orthogonal basis in the Krylov subspace $\mathcal{K}_k(A, r_0)$.

The notation $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ means that there exists certain real $\gamma_j,\ j = 0 : k-1$ such that

$$x_k = x_0 + \sum_{j=0}^{k-1} \gamma_j A^j r_0, \tag{4.28}$$

and this writes again $x_k = x_0 + p_{k-1}(A) r_0$. Since by definition

$$r_k = b - A x_k = b - A(x_0 + p_{k-1}(A) r_0) = (1 - p_k(A)) r_0$$

and $p_0 = r_0$, one can, using (4.28), reformulate the residual vector relation in its polynomial form:

$$r_k = \phi_k(A) r_0, \tag{4.29}$$

where

$$\phi_k(A) = 1 - p_k(A) \tag{4.30}$$

is a polynomial of degree not greater than k and that verifies $\phi_k(0) = 1$. This means we may impose the constraint that fixes the constant term of the residual polynomial to 1 and this guaranties the convergence of the Neumann series in A. Similarly (see for example Strako**v***s* in [63]), the k^{th} error can be written as a polynomial in A applied to the initial error. That is,

$$x^* - x_k = \phi_k(A)(x^* - x_0). \tag{4.31}$$

The relation (4.29) links the search of orthogonal residual vectors in Krylov subspaces to search of orthogonal polynomials in polynomial spaces. For this last, the orthogonality is related to the inner product defined by the Riemann-Stieltjes integral (see annexe C) with appropriate weight functions $\omega(\lambda)$ that play an important role in the convergence rate of the CG algorithm.

In the Krylov subspace, to get an optimal approximation of the solution at iteration k, the vector x_k is expected to be a minimizer of (4.31) that lies in the whole Krylov subspace of degree k. However, due to rounding errors, one may get an approximation polynomial of degree less than k. In this case, the approximation is said to be suboptimal. Otherwise, it is known that, the Krylov subspace

$$\mathcal{K}_k(A, r_0) = span\left\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\right\},$$

with the basis

$$\left\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\right\}, \tag{4.32}$$

is in general ill conditioned. Hence in practice, accumulation of rounding errors will certainly lead to loss of orthogonality [63]. In particular, for increasing k, the vectors

$A^j r_0,\ 1 \leq j \leq k-1$ will eventually become linearly dependent and this leads, as we said above, to a suboptimal solution because the computed direction lies in a Krylov subspace of order less than expected. Therefore, the arithmetic behaviour of the CG algorithm is often related to rounding errors due to loss of orthogonality among computed directions and residuals [45].

Let us now address the convergence rate of the CG algorithm at iteration k in a polynomials space. In this case, we have to minimize

$$\|x^* - x_k\|_A^2 = \min_{z \in x_0 + \mathcal{K}_k(A, r_0)} \|x^* - z\|_A^2.$$

where $\mathcal{K}_k(A, r_0)$ is a space of polynomials of degree not greater than k. Now, the problem becomes a search of a polynomial z. Thus, it can be verified that for any $z \in \mathcal{K}_k(A, r_0)$, we have that, $z = x_0 + \sum_{j=0}^{k-1} \gamma_j A^j r_0$, where $\gamma_j, \quad j = 0 : k-1$ are some real coefficients.

Let Φ_k denote the set of all polynomials of a degree not greater than k verifying $\phi_k(0) = 1$. The global minimizer $z = x_k$ of $\|x^* - x_k\|_A^2$ can be understood as a solution of

$$\min_{z \in x_0 + \mathcal{K}_k(A, r_0)} \|x^* - z\|_A^2 = \min_{\phi_k \in \Phi_k} (r_0^T \phi_k(A) A^{-1} \phi_k(A) r_0),$$

with $\phi(0) = 1$, for all $\phi \in \Phi_k$.

As the eigen decompositions

$$A = U \Lambda U^T \quad \text{and} \quad A^{-1} = U \Lambda^{-1} U^T \tag{4.33}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$ are allowed, we can write

$$\phi_k(A) = U \phi_k(\Lambda) U^T = U diag(\phi_k(\lambda_j)) U^T,\ j = 1, 2 \ldots, N = \sum_{j=1}^{N} \phi_k(\lambda_j) u_j u_j^T.$$

Thus

$$\|x^* - x_k\|_A^2 = \min_{\phi_k \in \Phi_k} \left\{ \sum_{j=1}^{N} \frac{(r_0, u_j)^2 ([\phi_k(\lambda_j)]^2)}{\lambda_j} \right\} \tag{4.34}$$

The relation (4.34) shows that, for Symmetric Positive Definite matrices, the CG rate of convergence is determined by the distribution of eigenvalues of the coefficients matrix and the components of r_0 in each eigenvector direction. This is particularly important from linear systems from MIRP since the initial residual contains the information about the images and often presents significant variability as we will indicate later in this chapter.

In addition, setting $y = r_0^T U$, we get

$$\|x^* - x_k\|_A^2 = \min_{\phi_k \in \Phi_k} \left\{ \sum_{j=1}^{N} \frac{y([\phi_k(\lambda_j)]^2) y^T}{\lambda_j} \right\}$$

$$= \min_{\phi_k \in \Phi_k} \left\{ \sum_{j=1}^{N} \frac{[\phi_k(\lambda_j)]^2 y_j^2}{\lambda_j} \right\}$$

and for $k = 0$

$$\|x^* - x_0\|_A^2 = \sum_{j=1}^{N} \frac{[\phi_0(\lambda_j)]^2 y_j^2}{\lambda_j} = \sum_{j=1}^{N} \frac{y_j^2}{\lambda_j}. \tag{4.35}$$

Note that, since A is Symmetric Positive Definite, for all $x, y \in \mathbb{R}^N$,

$$x^T A y = x^T U \Lambda U^T y$$

By setting $\overline{x} = (Ux)$ and $\overline{y} = (Uy)$ one writes

$$x^T A y = 0 \quad \text{if and only if } \ \overline{x}^T \Lambda \overline{y} = 0$$

and

$$\sum_{j=1}^{N} \lambda_j \overline{x}_j^T \overline{y}_j = 0$$

The ratio $\dfrac{\|x^* - x_k\|_A^2}{\|x^* - x_0\|_A^2}$ shows that the CG convergence rate is an orthogonal polynomial in the eigenvalues of A.

To get a bound on this convergence rate, let us define

$$\beta(\lambda_1, \lambda_2, \ldots, \lambda_N) = \min_{\phi_k \in \Phi_k} \max_{\lambda_j \in \sigma(A)} |\phi_k(\lambda_j)|, \tag{4.36}$$

we get

$$\begin{aligned} \|x^* - x_k\|_A^2 &= \sum_{j=1}^{N} \frac{[\phi_k(\lambda_j)]^2) y_j^2}{\lambda_j} \\ &\leq \beta(\lambda_1, \lambda_2, \ldots, \lambda_N)^2 \sum_{j=1}^{N} \frac{y_j^2}{\lambda_j} \\ &= \beta(\lambda_1, \lambda_2, \ldots, \lambda_N)^2 \|x^* - x_0\|_A^2. \end{aligned} \tag{4.37}$$

This leads to

$$\frac{\|r_k\|_{A^{-1}}}{\|r_0\|_{A^{-1}}} = \frac{\sqrt{\|x^* - x_k\|_A}}{\sqrt{\|x^* - x_0\|_A}} \leq \beta(\lambda_1, \lambda_2, \ldots, \lambda_N). \tag{4.38}$$

By replacing the discrete set $\sigma(A)$ by a large continuous set $[a,\, b] = E \supset \sigma(A)$, the minimax problem in (4.36) is known to admit a unique solution (see for example [66]) given by

$$\phi_{k+1}(p) = T_{k+1}\left(\frac{b + a - 2p}{b - a}\right) / T_{k+1}\left(\frac{b + a}{b - a}\right), \quad \forall,\, p \in [a, b]. \tag{4.39}$$

where $T_k, k = 0, 1, \ldots$ are shifted and scaled Tchebychev polynomials of first kind with $T_0(p) = 1, T_1(p) = p$ and $T_{k+1}(p) = 2pT_k - T_{k-1}$.

The advantage of this formula is the possibility to obtain orthogonal polynomials with a recurrence formulae. The CG algorithm is expected to provide a sequence of orthogonal residuals via the Lanczos process but the task is then to preserve this orthogonality in practical finite precision [2].

Let us link the Lanczos process to the CG algorithm. Given the matrix A and the initial residual r_0, the Lanczos process, in an ideal case, produces a sequence of

2. Note that the Lanczos process may also suffer from the same effects (loss of orthogonality)

orthonormal vectors $v_1, v_2, \ldots,$ via the recurrence relations (see [63, p.59]) that begin by

$$v_1 = \frac{r_0}{\|r_0\|},\ \delta_1 = 0$$

and for $k = 1, 2, \ldots,$

$$\begin{array}{lcl} \sigma_k & = & v_k^T(Av_k - \delta_k v_{k-1}), \\ w_k & = & Av_k - \sigma_k v_k - \delta_k v_{k-1}), \\ \delta_{k+1} & = & \|w_k\|,\ (\text{if } \delta_{k+1} = 0 \quad \text{then stop;} \\ v_{k+1} & = & \dfrac{w_k}{\delta_{k+1}} \end{array} \tag{4.40}$$

This process provides the matrix

$$V_k = [v_1, v_2, \ldots, v_k]$$

of size $N \times k$ formed by the Lanczos vectors and the tridiagonal matrix

$$L_k = \begin{pmatrix} \sigma_1 & \delta_2 & & \\ \delta_2 & \sigma_2 & \ddots & \\ & \ddots & \ddots & \delta_k \\ & & \delta_k & \sigma_k \end{pmatrix}$$

with positive subdiagonals.

Lanczos methods approximate k eigenvalues of the matrix A from those of the tridiagonal matrix L_k. The link between the CG and the Lanczos processes (see for example [64] include the relations

$$\left\{ \begin{array}{lcl} \sigma_k & = & \dfrac{1}{\alpha_{k-1}} + \dfrac{\beta_{k-1}}{\alpha_{k-2}},\ \beta_0 = 1,\ \alpha_{-1} = 1. \\ \delta_{k+1} & = & \dfrac{\sqrt{\beta_k}}{\alpha_{k-1}}, \\ v_{k+1} & = & (-1)^k \dfrac{r_k}{\|r_k\|} \end{array} \right. \tag{4.41}$$

where the reals α_k, β_k are from the CG algorithm 2.2 while the σ_k, δ_k are from the lanczos process 4.40. This allows to write

$$AV_k = V_k L_k + \delta_{k+1} v_{k+1} e_k^T, \tag{4.42}$$

where $V_k = [v_1, v_2, \ldots, v_k]$ is a matrix of Lanczos vectors and e_k^T is the k^{th} column of the N by N identity matrix. From $x_k = x_0 + p_k(A)r_0$, we may be interested by finding a vector y_k, via the use of orthogonality of the vectors r_k on the basis $\{v_1, v_2, \ldots, v_k\}$ of the subspace $\mathcal{K}_k(A, r_0)$. This means, the best approximation x_k at iteration k is such that

$$x_k = x_0 + V_k y_k. \tag{4.43}$$

To find y_k, notice that the condition on orthogonality of v_{k+1} on $v_1, v_2, \ldots, v_k$ allows to write

$$v_{k+1} = \phi_k(A)v_1$$

where $\phi_k(A) \in \Phi_k$ is an orthogonal monic polynomial (see Annexe (C) and the polynomial $\phi_k(A)$ can be determined by solving the problem at the right-hand of (4.36).

The obtained sequence of monic polynomials $1, \phi_1, \phi_2, \ldots$ that are orthogonals with respect to the weighted scalar product $\langle ., . \rangle_\omega$ defined by

$$\langle f, g \rangle_\omega = \int_a^b f(\lambda) g(\lambda) d\omega(\lambda), \quad \forall\, f, g \in \mathcal{P} \tag{4.44}$$

where $\mathcal{P}$ is the space of polynomials. The distribution function $\omega(\lambda)$ defined at finite points of increase $\lambda_1 < \lambda_2 < \ldots < \lambda_N$ (see Annexe C) corresponding to eigenvalues of the matrix A defines the weights. The corresponding Riemann-Stieltjes integral satisfies

$$\int_a^b f(\lambda) d\omega(\lambda) = \sum_{i=1}^{N} \omega_i f(\lambda_i)$$

can be solved accurately by Gauss quadrature.

In addition, the orthogonality of r_k with respect to vectors in V_k allows us to write

$$V_k^T r_k = 0$$
$$V_k^T (b - Ax_k) = 0,$$
$$V_k^T (b - Ax_0 - AV_k y_k)$$

such that

$$V_k^T A V_k y_k = V_k^T r_0 \tag{4.45}$$

Since $r_0 = \|r_0\| v_1$ and V_k^T constitute orthonormal vectors, The equation (4.45) writes

$$L_k y_k = \|r_0\| e_1, \tag{4.46}$$

where e_1 is the first column of the identity matrix and $L_k = V_k^T A V_k$ is the tridiagonal matrix constructed by the Lanczos process. By denoting the eigen decomposition of L_k by

$$L_k = S \Theta S^T = \sum_{j=1}^{k} \theta_k^j s_k^j {s_k^j}^T, \tag{4.47}$$

we can redefine the constant distribution function $\omega(\lambda)$ with the new k points of increase $a < \theta_k^1 < \theta_k^2, \ldots, \theta_k^k < b$

$$\omega_k^j = (e_1, s_k^j)^2, \quad \sum_{j=1}^{N} \omega_k^j = 1, \tag{4.48}$$

where s_k^j are eigenvectors of L_k and θ_k^j their associated eigenvalues. Thus, we can determine the k first polynomials from $\{1, \phi_1, \phi_2, \ldots, \phi_N\}$ with the associated Riemann-Stieltjes integral with the above distribution function by

$$\phi_l = \underset{\phi \in \Phi_l}{\operatorname{argmin}} \left\{ \int_a^b [\phi(\lambda)]^2 d\omega^k(\lambda) \right\}, \quad l = 0, 1, \ldots, k. \tag{4.49}$$

This is provided by the k^{th} Gauss quadrature approximation. Since the Gauss quadrature with k nodes and k weights can achieve $2k - 1$ degree of exactness, this ensures that the Riemann-Stieltjes integral will be exact for any polynomial of degree less or equal to $2k - 1$.

From the informations above, it is clear that a CG algorithm is stable and behaves correctly as expected when the process guaranties the preservation of orthogonality of the residuals, the A-conjugancy of the directions and linear independency of Krylov subspace basis.

However, this orthogonality and stability is related to rounding errors. To understand the effects of rounding errors, the work of Paige is well known in litterature (see [64], [65], [67]). Here we present, without demonstration some of the results wich allows us to interpret some numerical results in (4.6) and we reffer to papers [64] and [65] for more details.

Let us define the Ritz pairs from the Lanczos process. First, note that, the eigenpairs (θ_k^j, s_k^j) from (4.47) are called *Lanczos pairs* or primitive Ritz pairs. The *Ritz pairs* are

$$(\theta_k^j, y_k^j), \text{ such that } y_k^j = V_k s_k^j \tag{4.50}$$

where θ_k^j is called Ritz value while y_k^j is called Ritz vector. Note that, at iteration k, the Lanczos process approximates the Ritz pairs and one can verify from 4.42 that

$$Ay_k^j = \theta_k^j y_k^j + \delta_k v_{k+1}(e_k^T s_k^j). \tag{4.51}$$

The residual of the Ritz pair has 2-norm equal to

$$\|(e_k^T s_k^j)\delta_{k+1} v_{k+1}\|_2 = (e_k^T s_k^j)\delta_k. \tag{4.52}$$

Thus, in finite precision, Lanczos vectors lose orthogonality in the computation of r_k due to cancelation when δ_k is small. In this case, the quality of Ritz value is deteriorated. Three consequences of the relation (4.51) are of interest.

First, when (4.52) is too small, then a Ritz value has converged to an eigenvalue of the matrix A.

Second, the loss of orthogonality is observeed only in the direction of converged Ritz vectors.

Third, in finite precision, the loss of orthogonality in the Lanczos process depends on the matrix A and on the initial vector v_1. From this consequence, it is stated in [64, p.37] that, the only initial vector that may suppress the loss of orthogonality should be an initial vector where the order of magnitude of the components vary significantly.

The statement above explains likely why for the same system matrix, we may have different convergence rate for different images, even with the same size. Especially because the order of magnitude may vary significantly from one image to another depending on the acquisition mode and the regularization.

Since we are able to act on the initial vector, and at the same time, act on the coefficient matrix we may expect high rate of convergence using coefficient matrix with well suited spectrum that may be less sensitive to rounding errors. This is allowed by preconditioning techniques. In addition, it enables a high rate of convergence obtained if the spectrum becomes more clustered. In the following section, we address preconditioning techniques that allow clustering the spectrum of the system matrix.

4.2.2 Preconditioning the CG algorithm

As mentioned in Section 2.2.3, preconditioning a linear system (4.1) aims to transform it into an equivalent one. The new linear system is expected to have more favorable properties concerning its eigenvalues distribution or clusters. To be effective,

the preconditioner should be cheap to construct, to store and to apply. Below we present three ways of transforming the linear system when preconditioning it.

Left preconditioning

The basic idea of this technique is to design the preconditioner P that approximates A and is easy to invert. Then, one may compute $P^{-1} \simeq A^{-1}$, that approximates the inverse of A, and efficiently solve the system

$$P^{-1}Ax = P^{-1}b. \tag{4.53}$$

A major drawback of the system (4.53) is that it may lose the symmetry because the symmetry of A and P does not imply symmetry of $P^{-1}A$. In addition, the inverse may not be easy to compute, to store or to apply. However, for P positive definite, $P^{-1}A$ is symmetric with respect to the P-inner-product $\langle ., . \rangle_P$.

Right preconditioning

As can be seen in the left preconditioning technique above (4.53), the preconditioner affects both the operator A and the right-hand side b. The right preconditioning affects only the operator A but not b using a variable change. However, it still loses symmetry. It writes

$$AP^{-1}u = b, \quad x = P^{-1}u. \tag{4.54}$$

A compromise between left and right preconditioning is the use of split preconditioning.

Split preconditioning

To preserve the symmetry of the linear system while affecting less the right-hand side, the split preconditioning technique is used. In this technique, the preconditioner is used in factored form $P = LL^T$ where L may be the Cholesky factor or a matrix square root of A. Then, the linear system writes

$$L^{-1}AL^{-T}u = L^{-1}b \qquad \text{where} \qquad L^T x = u. \tag{4.55}$$

The convergence rate of the CG algorithm depends now on $k(L^{-1}AL^{-T})$ rather than on $k(A)$. This is done implicitly done in the PCG algorithm.

Without mentioning the physics-based preconditioners, multigrid or multilevel preconditioners and block-preconditioners, there are two main classes of algebraic preconditioners: implicit and explicit preconditioners. They are qualified as algebraic since they are based only on information contained in the coefficient matrix of the system. Implicit preconditioners address the problem by approximating the matrix A with a matrix that is easy to invert, while explicit preconditioners approximate the inverse of A.

Figure 4.3 presents a non exhaustive taxonomy on classes of these preconditititioners for SPD linear systems. Below we present a brief description of each of them.

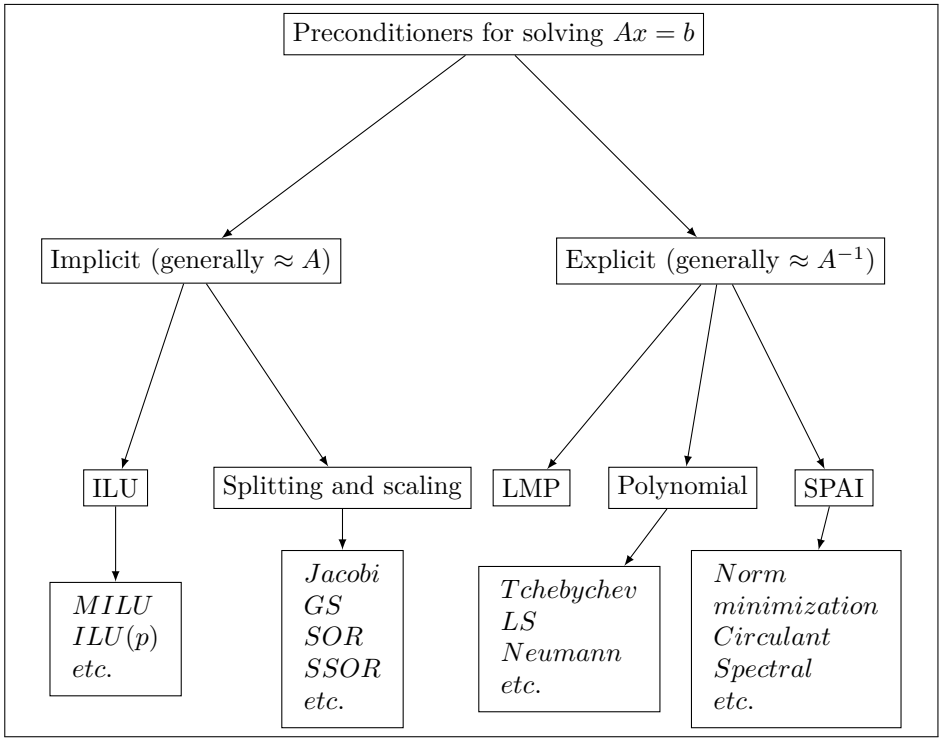


Figure 4.3: Taxonomy on preconditioners of SPD systems

4.2.3 Incomplete LU preconditioning methods

The Incomplete LU factorization (ILU) of the matrix A gives rise to the denoted *ILU-preconditioner*, one of the most effective preconditioners for large sparse linear systems. It is used to approximate a lower sparse triangular matrix L and an upper sparse matrix U such that LU aproximates A. In the symmetric case, it reduces to incomplete Cholesky factorization ILL^T and the cost may be halved, compared to ILU for the nonsymmetric case. However, this technique may lose favorable structure as sparsity of the initial matrix. To keep a favorable structure, a variety of ILU preconditioning methods have been developed. For example, some keep non-zero elements only in predetermined positions. They include the zero fill-in ILU ($ILU(0)$), the p-level fill-in ILU ($ILU(p)$) and the ILU with threshold ($ILUT$). Other ILU methods use to substract entries of the newly formed row or column from the main diagonal to preserve the column sum. They are called Modified ILU ($MILU$). More on these preconditioners and their variants can be found in [68]. Preconditioners based on incomplete factorization may provide ill-conditioned factors and lead to inaccurate solutions. In addition, these methods are still highly sequential and may not present any advantage on parallel computers [45]. Splitting preconditioners present an interesting alternative.

4.2.4 Splitting preconditioning methods

From the linear system $Ax = b$ and given an initial solution x_0, splitting preconditioning techniques approximate the current solution x_{k+1} from the previous x_k based on the iterations

$$Mx_{k+1} = Nx_k + b, \tag{4.56}$$

where the splitting of A is given by

$$A = M - N. \tag{4.57}$$

When splitting the matrix A, it is important to ensure that M is regular and easy to invert for effectiveness. This gives rise to iterates

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b. \tag{4.58}$$

The splitting preconditioning methods are, in general, highly parallel and generally easy to implement [69]. In most cases, the splitting is considered as the sum of three components of A, setting

$$A = D - L - U, \tag{4.59}$$

where D, L, U are respectively the diagonal, the strict lower triangular and the strict upper triangular components of A. Since for symmetric positive definite matrices, the diagonal D is regular, the following preconditioning techniques can be defined (see [68, pp.283-284]).

Jacobi preconditioning method

Set $M = D,\ N = L + U$ and get the iterations (4.58). For components of the vector x_{k+1}, the i^{th} component writes (see [70, 119])

$$x_{k+1}^i = -\sum_{j=1\ j\neq i}^{N} \frac{a_{ij}}{a_{ii}} x_k^j + \frac{b_i}{a_{ii}}. \tag{4.60}$$

This possibility of computing components independently each other makes the method appropriate for parallel computing.

Gauss-Seidel method (GS)

Given the splitting (4.57), set $M = D - L,\ N = U$ and get the iterations (4.58). For components of the vector x_{k+1}, the i^{th} component writes (see [70, 120])

$$x_{k+1}^i = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_{k+1}^j - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_k^j + \frac{b_i}{a_{ii}}. \tag{4.61}$$

Observe that the Gauss-Seidel method is like Jacobi method except that Gauss-Seidel method uses the $i-1$ components computed yet in the current approximation of the other components. Setting $M = D - L$ gives the so called forward Gauss-Seidel method that is equivalent to the backward Gauss-Seidel method obtained by setting $M = D - U$.

Successive Overrelaxation method (SOR)

The SOR is a variant of GS where a relaxation parameter $\omega \in \mathbb{R}$ allows to set $M = \frac{1}{\omega}(D - \omega L),\ N = \frac{1}{\omega}((1-\omega)D + \omega U)$, where $\omega \in \mathbb{R}$ is a relaxation parameter, and get the iterations (4.58). For components of the vector x_{k+1}, the i^{th} component writes (see [70, 120])

$$x_{k+1}^i = x_k^i + \omega \left(-\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_{k+1}^j - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_k^j + \frac{b_i}{a_{ii}} - x_k^i \right).$$

Setting $\omega = 1$, one gets the Gauss-Seidel method. Thus one can define the forward and backward SOR following the definition in the GS method. The Symmetric SOR (SSOR) uses simultaneously the forward and backward framework.

4.2.5 Sparse Approximation Inverse (SPAI) methods

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{N \times N}$ is sparse and large. The SPAI methods approximate the preconditioner M that minimizes the Frobenius norm $\|I - AM\|$, where I is the identity matrix. that is

$$\min_{M \in S} \|I - AM\|_F^2 = \min_{M \in S} \sum_{j=1}^{N} \|e_j - Am_j\|_2^2,$$

where S is the set of sparse matrices and $e_j,\ m_j$ are the j^{th} columns of, respectively, the idenity matrix and the matrix M. The importance of this constraint S is to filter the entries of A^{-1} with respect to their contribution quality for the preconditioner. For example, one may need to drop entries that are small in absolute value with respect to a given threshold. For general sparse matrices, it is not known in advance where large entries of A^{-1} are located. But for banded SPD matrices, it is known that the entries of A^{-1} are bounded along each row or column and exponentially decaying [45]. Thus, the inverses of such matrices can be approximated by low-cost techniques.

Different ways of approximating these inverses lead to different preconditioners. Among the most known invariants of th SPAI method, is the factorized sparse inverse approximation and the incomplete factorization followed by the inverse approximation.

In SPAI methods, the sparsity patterns may be fixed in advance. In this case, one reduces the search to the submatrix $\hat{A} = A(I, J)$ where I and J denote respectively the particular rows and the columns indices of A that enter the definition of e_j and m_j. One gets smaller vectors $\hat{e}_j$ and $\hat{m}_j$ such that the problem reduces to a small least squares problem where the norm

$$\|\hat{e}_j - \hat{A}\hat{m}_j\|_2$$

is minimized. There are several strategies to capture automatically the sparsity patterns when it is difficult to prescribe it in advance. These strategies may use geometric and topological informations or use information on the absolute values of the entries.

An attractive feature of SPAI method is that they are naturally parallelizable. However, cheaper preconditioners are those with high sparsity patterns but these

may not lead to significant improvement in the solver. More improvement may be obtained when the preconditioner becomes dense but this also becomes expensive. A smart compromise is required to use SPAI preconditioners. In the following section, we focus on the Limited Memory Preconditioners (LMP) that can be seen as low cost explicit preconditioner.

4.3 Limited Memory Preconditioners (LMP)

In general, the LMP preconditioner approximates the inverse of a matrix within a subspace, using a reduced number of vectors. Let the matrix $A \in \mathbb{R}^{N\times N}$ be SPD and S be a matrix of size $N \times (k+1)$ containing $(k+1)$ linearly independent vectors $s_0, s_1, \ldots, s_k$. Following J. Tshimanga in [71], the LMP preconditioner is the matrix

$$H_{k+1} := \left[I_N - S(S^TAS)^{-1}S^TA\right] M \left[I_N - AS(S^TAS)^{-1}S^T\right] + \theta S(S^TAS)^{-1}S^T, \tag{4.62}$$

where M is another preconditioner called first-order preconditioner and the scalar $\theta \in \mathbb{R}$ is a shift factor. Here we consider $M = I$, the identity matrix, and $\theta = 1$.

One can verify that, for $k+1 = N$, we have $H_N = A^{-1}$. Observe that, for k small enough compared to N, the matrix S^TAS is small and consequently easy to inverse. Thus, the most expensive operation in this construction would be the product AS or the equivalent S^TA. But note that, one does not need to compute or explicitly store entries of H_{k+1} but simply use matrix-vector products in some procedures both in construction and application. The needed information concerns the vectors $s_j,\ j = 0, 1, \ldots, k$ and some information about the matrix S^TAS of size $k+1 \times k+1$. Note that the preconditioner H_{k+1} is invariant under change of basis for vectors in S. Thus, the following proposition is proved in [71].

Proposition 4.4

Let the matrix $Z \in \mathbb{R}^{N\times(k+1)}$ verify $Z = SX$ where X is a square and invertible matrix of order $k+1$. Then

$$S(S^TAS)^{-1}S^T = Z(Z^TAZ)^{-1}Z^T,$$

and the preconditioner H_{k+1} in (4.62) writes again

$$H_{k+1} = \left[I_N - Z(Z^TAZ)^{-1}Z^TA\right]\left[I_N - AZ(Z^TAZ)^{-1}Z^T\right] + Z(Z^TAZ)^{-1}Z^T.$$

For the preconditioner H_{k+1} to approximates the inverse of A, the application of the preconditoner may take various forms. Thus, the proposition below presents different forms of the LMP H_{k+1}. These forms include the inverse B_{k+1} of H_{k+1} and the factors that allow the LMP to be used in a splitting preconditioning technique.

Proposition 4.5

Assume H_{k+1} is constructed following (4.62), then, these equalities hold

$$\begin{aligned} H_{k+1} &= (I_N - SR^{-1}R^{-T}S^TA)(I_n - ASR^{-1}R^{-T}S^T) + (SR^{-1}R^{-T}S^T), \\ B_{k+1} &= I_N + ASR^{-1}R^{-T}S^TA - SU^{-1}U^{-T}S^T, \qquad (4.63) \\ G_{k+1} &= I_N - SR^{-1}R^{-T}S^TA + SR^{-1}U^{-T}S^T, \qquad (4.64) \\ G_{k+1}^{-1} &= I_N - SU^{-1}U^{-T}S^T + SU^{-1}R^{-T}S^TA, \qquad (4.65) \end{aligned}$$

with $B_{k+1}^{-1} = H_{k+1}$ and $G_{k+1}G_{k+1}^T = H_{k+1}$ and where R and U are upper triangular matrices from Cholesky decomposition of $S^TAS = R^TR$ and $S^TS = U^TU$, respectively.

Proof. See [71, p.50]. □

This theorem allows right, left or split preconditiong techniques to be used with an LMP. In addition, if the Cholesky factors R and U are available, the computation efforts may be halved. Properties of the preconditioner H_{k+1} have been studied in [71] and [72]. Some of these properties are:

First, the preconditioner H_{k+1} may act on one part of the spectrum of A and leave the other part invariant. In practice, one should keep the part that already has favorable properties such as a cluster of eigenvalues arround 1 invariant. Second, one can verify that H_{k+1} amounts to A^{-1} if $k+1 = N$. Note that in this case, S is a square matrix of full rank and thus invertible. Third, the application of H_{k+1} to A may extend the pre-existing cluster of eigenvalues around 1 without either destroying the pre-existing ones or extending pre-existing clusters around other eigenvalues.

As stated in the paper [72], about the cluster of eigenvalues around 1 the remark below expresses what we have observed by simulations.

Remark 4.2

Consider that the matrix A has a cluster of eigenvalues arround 1 of multiplicity m with $0 \leq m < N$, and associated to eigenvectors $\{v_1, \ldots, v_m\}$. Assume in addition that, among the chosen vectors $[s_0, \ldots, s_k]$ to construct the preconditioner, there are k' with $0 \leq k' \leq k+1$ that are a linear combination of vectors from V. Thus, the multiplicity of the cluster around 1 will amount to $m + (k+1) - k'$. Thus, in the best case, the maximal cluster arround 1 is then $m+k+1$ obtained for $k' = 0$. This means all the used vectors s_j are linearly independant with the eigenvectors associated to the eigenvalue 1. In general, two cases can be distinguished.

1. *If $0 \leq m < k+1$ (more vectors in S than the dimension of the eigenspace associated to the eigenvalue 1) or $m > k+1$ while $k' < k+1$ (less vectors in S than the dimension of the eigenspace associated to the eigenvalue 1 but some of them are not linear combination of eigenvectors $v_1, \ldots, v_m$), in both cases the dimension of the eigenspace associated to the eigenvalue 1 is increased (this is additivity). This means the cluster around 1 is increased but H_{k+1} will act on the spectrum of A as $H_{k-k'+1}$.*
2. *If $m > k+1$ and $k' = k+1$, one obtains invariance, that is the cluster arround 1 remains m. H_{k+1} will act on the spectrum of A as $H_0 = I_N$ the identity matrix. This is the worst case.*

From this remark, one is guaranteed that the LMP preconditioner can not deteriorate the spectrum. However, it advises a wise choice of vectors in S (that are not in the eigenspace associated to eigenvalue 1). Observe that, no assumption is made in advance on the vectors in S apart from their linear independency. However, it is clear from the remark above that, the more one chooses vectors with specific properties, such as orthogonality, A-conjugancy or A-invariance, the more the preconditioner is effective in the CG-algorithm. Thus, the quality of the preconditioner H_{k+1} depends on properties of the vectors used for its construction and their relationship with respect to the matrix A. Three points below assume that the vectors in S have some specific properties.

Using orthogonal vectors

Let vectors $s_0, s_1, \ldots, s_k$ be orthogonal. That is

$$s_j^T s_k \begin{cases} = 0 & \text{if} \quad j \neq k, \\ = 1 & \text{if} \quad j = k. \end{cases} \tag{4.66}$$

Since these orthogonal vectors have no relationship with the matrix A, they may not allow significant advantages for the LMP H_{k+1}. However this orthogonality property saves cost and simplifies the computation of B_{k+1} and G_{k+1}^{-1} where $S^T S$ occurs (see 4.65), because $S^T S$ amounts to the identity I_{k+1} in this case.

Using A-invariant subspace basis vectors

In opposition to simple orthogonality property, one gets more simplification by considering that $s_0, \ldots, s_k$ are in a A-invariant vectors subspace. This means that, for $j = 0, 1, \ldots, k$,

$$As_j \in span(s_0, s_1, \ldots, s_k).$$

As a consequence, there exists a non-singular matrix X of order $k+1$ such that $AS = SX$. This leads to simplifications in the formulation of the LMP forms given in equations (4.65) as stated in the following proposition.

Proposition 4.6

Let $A \in \mathbb{R}^{N\times N}$ be SPD and assume that the columns in $S \in \mathbb{R}^{N\times(k+1)}$ form a basis of an A-invariant subspace, that is $AS = SX$ where X is an invertible matrix of order $k+1$. For H_{k+1} defined in (4.62), these equalities hold:

$$H_{k+1} = I_N - SU^{-1}U^{-T}S^T + SX^{-1}U^{-1}U^{-T}S^T, \tag{4.67}$$

$$B_{k+1} = I_N + SU^{-1}U^{-T}X^T S^T - SU^{-1}U^{-T}S^T, \tag{4.68}$$

$$G_{k+1} = I_N - SR^{-1}R^{-T}X^T S^T + SR^{-1}U^{-T}S^T, \tag{4.69}$$

$$G_{k+1}^{-1} = I_N - SU^{-1}U^T S^T + SU^{-1}R^{-T}X^T S^T, \tag{4.70}$$

where R and U are upper triangular matrices from Cholesky factorisation such that $S^T AS = R^T R$ and $S^T S = U^T U$, while $B_{k+1}^{-1} = H_{k+1}$ and $G_{k+1}G_{k+1}^T = H_{k+1}$.

The main advantage of using A-invariant subspace of vectors is that, instead of computing the product AS with matrices of respective orders $N \times N$ and $N \times (k+1)$,

one may compute the product SX with matrices of respective orders $N \times (k+1)$ by $(k+1) \times (k+1)$.

Using A-conjugate vectors

A-conjugate vectors are defined by the relation

$$s_j^T A s_i \begin{cases} = 0 & \text{if} \quad j \neq i \\ > 0 & \text{if} \quad j = i, \end{cases} \tag{4.71}$$

for $j, i = 0, 1, \ldots$. With A-conjugate vectors in S, one gets significant simplification in the computation and the application of the preconditioner H_{k+1} as stated in Lemma 4.7 and Proposition 4.8 below.

Lemma 4.7

Let $A \in \mathbb{R}^{N \times N}$ be SPD and assume that the columns in $S \in \mathbb{R}^{N \times (k+1)}$ are conjugate with respect to A. Then

$$H_{k+1} = \left(I_N - \sum_{j=0}^{k} \frac{s_j s_j^T}{s_j^T A s_j} A \right) \left(I_N - \sum_{j=0}^{k} A \frac{s_j s_j^T}{s_j^T A s_j} \right) + \sum_{j=0}^{k} \frac{s_j s_j^T}{s_j^T A s_j}. \tag{4.72}$$

Proof. See [71, p. 69] □

Notice that this formulation allows one to avoid computing $s_j^T A s_k$ whenever $j \neq k$. In addition, it allows to use a recurrence formulation of different forms of the LMP as shown in the following proposition.

Proposition 4.8

Let assumptions of Lemma 4.7 be satisfied. Then, setting $H_0 = B_0 = I_N$ we have:

$$H_{k+1} = \left(I_N - \frac{s_k s_k^T}{s_k^T A s_k} A \right) H_k \left(I_N - A \frac{s_k s_k^T}{s_k^T A s_k} \right) + \frac{s_k s_k^T}{s_k^T A s_k}, \tag{4.73}$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{A s_k s_k^T A}{s_k^T A s_k}. \tag{4.74}$$

$$G_{k+1} = [I_N - s_k \left(\rho_k s_k^T A - \gamma_k s_k^T B_k \right)] G_k, \tag{4.75}$$

$$G_{k+1}^{-1} = G_{k+1}^{-1} [I_N + s_k \left(\gamma_k s_k^T A - \sigma_k s_k^T B_k \right)], \tag{4.76}$$

where $\rho_k = \left(s_k^T A s_k \right)^{-1}$, $\gamma_k = \left(s_k^T B_k s_k \right)^{-\frac{1}{2}} \left(s_k^T A s_k \right)^{-\frac{1}{2}}$ and $\sigma_k = \left(s_k^T B_k s_k \right)^{-1}$.

With respect to the above properties, three types of LMP preconditioners have been studied: spectral LMP, Ritz-LMP and Quasi-newton LMP. They are presented and compared in [71]. Here we simply analyze the application of the spectral LMP on our linear systems while the others are briefly mentioned.

The spectral LMP

The spectral LMP uses eigeninformation from the SPD matrix $A \in \mathbb{R}^{N \times N}$. Since eigenvectors verify *orthogonality, A-conjugacy and A-invariance*, they are well suited

to build the LMP preconditioner. This is the LMP we are interested in and that is developed roughly in the following section. However, one should note that it is difficult to obtain these eigen elements for general large sparse matrices. This is the reason why one may use one of the other following LMP preconditioners with respect to the context.

The Ritz-LMP

Let $A \in \mathbb{R}^{N\times N}$ be SPD. Approximating eigenpairs of A from a subspace $S \subset \mathbb{R}^N$ leads to the Ritz pair (4.50) $(\theta, z) \in \mathbb{R} \times S$ and a residual vector (4.51)

$$r = Az - \theta z.$$

The Ritz-LMP builds S from this Ritz information (Ritz pairs), which is an approximation of the spectral information. J. Tshimanga in [71] showed that in many cases, the Ritz-LMP behaves like the spectral LMP. However, the Ritz pairs may be also expensive to compute. Then, the quasi-Newton LMP presented below is an alternative.

The quasi-Newton LMP

Instead of using spectral information (spectral LMP) or their approximation (Ritz-LMP), the quasi-Newton LMP builds S using A-conjugate vectors from the CG algorithm. These are the directions $p_j,\ j = 1 : k+1$ that verify the relation (2.30). It supposes one has to call first the CG algorithm without preconditioner (or with first-level preconditioner) such that information is gathered and then build the LMP preconditioner with this information. This LMP will then be used in the next call of the CG algorithm. The need to run first iterations with a non-preconditioned CG makes the quasi-Newton LMP less attractive for very large and ill-conditioned systems than other preconditioning techniques.

4.3.1 The spectral LMP on A_{dif}

Since one can compute some eigenvalues of A_{dif} in a relatively easy way, (see (4.20)), the spectral LMP has to be investigated. Let us recall that eigen elements $(\lambda_j, v_j)_{j=1:N}$ verify the properties

$$\begin{array}{lcl} Av_i & = & \lambda_i v_i, \quad v_i^T v_i = 1, \ \ i = 1 : N \\ v_i^T v_j & = & 0, \quad \text{if} \quad i \neq j, \ \ i, j = 1 : N \\ v_i^T A v_j & = & 0, \quad \text{if} \quad i \neq j, \ \ i, j = 1 : N \\ v_i^T A v_i & = & \lambda_i > 0, \ \ i = 1 : N \end{array} .$$

When $k+1$ eigenvectors are available, then the LMP preconditioner is constructed in one of the formulations presented in the proposition below from [71, p.75]

Proposition 4.9

Let some eigenvectors $v_0, v_1 \ldots, v_k$ of A be given. Then, we have

$$H_{k+1} = \prod_{j=0}^{k} \left[I_N - \left(1 - \frac{1}{\lambda_j}\right) v_j v_j^T \right], \tag{4.77}$$

$$G_{k+1} = \prod_{j=0}^{k} \left[I_N - \left(1 - \frac{1}{\sqrt{\lambda_j}}\right) v_j v_j^T \right], \tag{4.78}$$

$$G_{k+1}^{-1} = \prod_{j=k}^{0} \left[I_N - \left(1 - \sqrt{\lambda_j}\right) v_j v_j^T \right], \tag{4.79}$$

$$B_{k+1} = \prod_{j=k}^{0} \left[I_N - (1 - \lambda_j) v_j v_j^T \right]. \tag{4.80}$$

The most important property here is this possibility of using the approximation recursively. This helps to use the preconditioner with a chosen number of eigenvectors and to improve it later if needed. Observe that when applying H_{k+1}, it is needed to store $2(k+1)$ vectors of $\mathbb{R}^N$ (see [72, p.17]). In addition, the application to an arbitrary vector y costs $10(k+1)N$ operations [71, p.53]. This indicates that one should avoid increasing the number $k+1$ of vectors in S. At the same time, this number indicates the number of eigenvalues that will amount to one in the preconditioned system. So once more, a compromise is needed.

Figure 4.4 and Figure 4.5 visualize the effects of spectral LMP on the spectrum of A_{dif}. On a matrix of size (512×512), the spectrum of A is shown (blue) and the spectrum of $H_{k+1}A$ (red) is shown for $k+1 = 40$ and $k+1 = 240$ in Figure 4.4 while $k+1 = 360$ and $k+1 = 512$ in Figure 4.5.

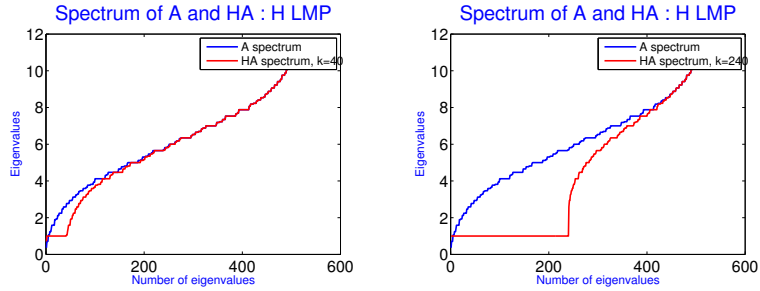


Figure 4.4: Comparison of the spectrum of $A = A_{dif}$ and $H_{k+1}A$. $n_1 = n_2 = n_3 = 8$, $N = 512$. The spectrum of A is shown (blue) and the spectrum of $H_{k+1}A$ (red) is shown for $k+1 = 40$ and $k+1 = 512$ respectively.

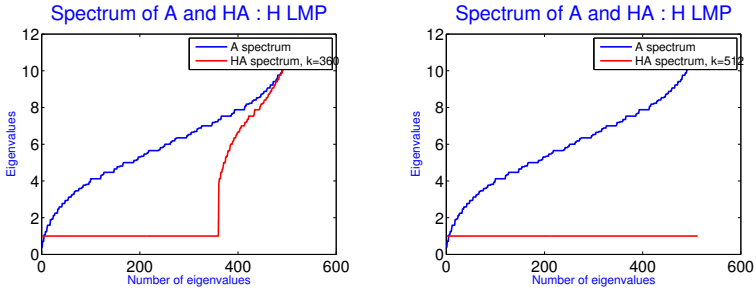


Figure 4.5: Comparison of the spectrum of $A = A_{dif}$ and $H_{k+1}A$. $n_1 = n_2 = n_3 = 8$, $N = 512$. The spectrum of A is shown (blue) and the spectrum of $H_{k+1}A$ (red) is shown for $k+1 = 360$ and $k+1 = 512$ respectively. For $k+1 = N = 512$, $H_{k+1}A$ is the identity.

4.3.2 LMP limitation for MIRP

The LMP preconditioner uses a discrete scalar product. Using vectors from S will amount eigenvalues to 1, one after one. Even when the eigenvalues are quasi equal, each of them has to be treated since their related eigenvectors are different. Thus, when there are too many eigenvalues to treat, the LMP may take significant memory and computing time. For example, for systems whose spectrum are represented in Figure 4.4, there is almost only one eigenvalue around 1 before preconditioning. The LMP needs respectively $40, 240, 360$ and 512 vectors in S (see (4.62)) to build an LMP that increases the cluster around 1 to respectively $40, 240, 360$ and 512 eigenvalues. Then, for such very large and sparse systems, the storage and application of LMP become more expensive than other techniques such as polynomial or sparse factorization preconditioners.

Since the extreme eigenvalues can be known or well approximated for matrices A_{el} and A_{dif}, one way of dealing with this increasing need of storage and computation cost is to use a continuous scalar product. Such a scalar product may use some numerical quadrature with high degree of exactness. This is appropriate to polynomial preconditioners that may capture the spectrum with low-degree polynomials. The polynomial preconditioners are presented in the following section. For this purpose, we may exploite the use of polynomial preconditioners.

4.4 Polynomial preconditioning for MIRP

At first glance, polynomial preconditioners are less appropriate for large and sparse SPD linear systems solved using Krylov subspace methods. This can be stated by observing that, in the CG algorithm, the main goal is to reduce the number of matrix-vector products. Unfortunately, the polynomial preconditioners require also matrix-vector products in the preconditioning phase. Thus, to stay competitive, the number of iterations in the preconditioning phase must stay small while leading to significant improvement to the convergence rate of the CG. It has been established (see for example [73]) that, if m is the number of terms in the polynomial, the gen-

eral cost increases linearly in $\mathcal{O}(m+1)$ although the number of iterations in the CG decreases more slowly than $\mathcal{O}(1/(m+1))$. However, there exists some motivations to use polynomial preconditioners, within these presented below.

Motivation for polynomial preconditioner

There are features and context in which polynomial preconditioners are really effective and competitive. Here is a description of a specific context when they are still effective. First, in the context where at least, extreme eigenelements are explicitly known or their reliable approximation is available. Second, when large sparse matrices are SPD and that the whole spectrum can be captured by a low-degree polynomial [74]. Third, when distributed memory parallel machines are used where dot product operations and access to matrices need to be significantly reduced [45]. Fourth, when the spectrum is almost continuous, because in opposition to other Krylov methods, polynomial preconditioners are built using a continuous scalar product.

In addition, from [75, pp.46-47] consider m, the number of terms in the polynomial. It has been established that: "*k steps of polynomial preconditioning methods are exactly equivalent to km steps for classical original methods. For Tchebyshev methods, the preconditioned iteration may improve the number of iterations by at most a factor m. Thus, even if there is no gain in the number of matrix-vector products, an important fraction of inner-products and update operations are eliminated. Thus, efficiency is increased.*"

Furthermore, in addition to these criteria, one should note that, polynomial preconditioners do not need any additional storage and may be combined with other preconditioners. Two most known polynomial preconditioners are discussed in the following section and numerical results are presented.

4.4.1 Introduction to polynomial preconditioners

Let denote $\mathcal{P}_m = \left\{ P_m(\lambda) \equiv \sum_{i=0}^{m} c_i \lambda^i \mid c_i \in \mathbb{R} \right\}$, the set of polynomials of degree less or equal to m. A polynomial preconditioner transforms the linear system

$$Ax = b$$

in

$$P_m(A)Ax = P_m(A)b. \tag{4.81}$$

Following [76], the construction of the preconditioner can be achieved by searching the best polynomial, that is, a low degree polynomial allowing to approximate the inverse of the coefficient matrix A. This can be achieved by minimizing

$$\min_M \|I - p_m(A)A\|$$

in appropriate norm, and imposing the initial condition $p_0(A) = 1$ to the spectral radius is less or equal to 1. Let set $p_m(A) = M$, and use the eigen decomposition $A = U\Lambda U^{-1}$ where $\Lambda = diag(\lambda_1, \ldots, \lambda_N)$ as stated in (4.33). The above minimization problem is well suited in the infinity norm and writes

$$\text{find} \quad p_m \in \mathcal{P}_m \; such \; that \; \min_{p \in \mathcal{P}_m} \|I - p_m(A)A\|_\infty \tag{4.82}$$

Note that

$$\min_{p\in\mathcal{P}_m} \|I - p_m(A)A\|_\infty \leq \min_{p_m\in\mathcal{P}_m} \max_{\lambda\in\sigma(A)} |1 - p_m(\lambda)\lambda|$$

By setting

$$\phi_m = 1 - p_m(\lambda)$$

and Φ_m the set of polynomials ϕ_m as defined in (**??**) we have

$$\min_{p\in\mathcal{P}_m} \|I - p_m(A)A\|_\infty \leq \min_{\phi_{m+1}\in\Phi_{m+1}} \max_{\lambda\in\sigma(A)} |\phi_{m+1}(\lambda)|$$

where we recall from (4.27), (4.28) and (4.29) that

$$r_m = b - Ax_m, \tag{4.83}$$
$$x_m = x_0 + \phi_{m-1}(A)r_0, \tag{4.84}$$

where r_m is the m^{th} residual and from (4.30)

$$r_m = \phi_m(A)r_0. \tag{4.85}$$

The only difference with 2.24 is the continuous inner product (see e.g [69]) that must valid on the spectrum $\sigma(A)$. Therefore, polynomial Krylov iterative methods are often used when the spectrum is known or one may replace the discrete set of eigenvalues by a larger and continuous set [76, p.196].

Let $\lambda_{min}, \lambda_{max}$ be the smallest and the highest eigenvalues of the matrix A, respectively. One can consider

$$\Theta = [a, b] \subset \mathbb{R}, \tag{4.86}$$

an interval that contains $\sigma(A)$ with $0 \leq a \leq \lambda_{min} < \lambda_{max} \leq b$, to be the best approximation of $\sigma(A)$. Given the approximation of extreme eigenvalues, one may implement the polynomial Krylov iterative method following these two fundamental properties:

1. The k^{th} residual polynomial r_k is the one that minimizes the uniform norm among all residual polynomials in Θ.
2. Polynomials $\{r_k\}$ are orthogonal with respect to the weighted scalar product (4.44) defined on Θ.

Thus, the aim of the polynomial iterative methods is to guarantee that, at each iteration, the chosen polynomial $r_k(A)$ is such that

$$\|r_k\|_A \leq \min_{\phi_k\in\mathcal{P}_k, \phi_k(0)=1} \max_{\lambda\in\Theta} |\phi_k(\lambda)| \, \|r_0\|_A. \tag{4.87}$$

This means one looks for an approximation solution of the minmax problem

$$\min_{\phi_k\in\mathcal{P}_k, \phi_k(0)=1} \max_{\lambda\in\Theta} |\phi_k(\lambda)|. \tag{4.88}$$

One way of building such orthogonal residual polynomials is the use of the recurrence relationship (explained in Annexe C):

$$\begin{aligned} &\phi_{-1}(\lambda) = 0, \\ &\phi_0(\lambda) = 1, \\ &\gamma_k\phi_k(\lambda) = [\lambda + (\gamma_k + \beta_k)]\phi_{k-1}(\lambda) - \beta_k\phi_{k-2}(\lambda), \ k \geq 1. \end{aligned} \tag{4.89}$$

Below the Neumann and Tchebychev polynomial preconditioners are presented.

4.4.2 Neumann polynomial preconditioner

The Neumann preconditioner exploits the relation

$$\frac{1}{1-\lambda} = \sum_{i=0}^{\infty} \lambda^i, \ (\lambda \in \mathbb{R}), \tag{4.90}$$

for all $|\lambda| < 1$, that proves that the sequence $\sum_{i=0}^{\infty} G^i, \ \forall G \in \mathbb{R}^{N\times N}$ converges if and only if $\rho(G) < 1$ where $\rho(G)$ denotes the spectral radius [3] of G. Thus we have

$$\sum_{i=0}^{\infty} G^i = (I-G)^{-1}. \tag{4.91}$$

To show how this is used to aproximate the inverse of a SPD matrix $A \in \mathbb{R}^{N\times N}$, let us consider $G = (I - \omega A)$, where ω is a scaling factor chosen to ensure that $\rho(G) < 1$. Rewriting $\omega A = I - G$ one gets

$$A^{-1} = \omega(I-G)^{-1}. \tag{4.92}$$

Thus, the approximation of the inverse of A is allowed by the relation

$$A^{-1} = \omega(I-G)^{-1} = \omega \sum_{i=0}^{\infty} G^i \approx \omega(I + G + G^2 + \ldots + G^{m-1}), \tag{4.93}$$

where m is a natural integer. This lead to the sought Neumann preconditioner of the form

$$P_m(A) = \omega(I + G + G^2 + \ldots + G^{m-1}), \quad G = (I - \omega A). \tag{4.94}$$

For SPD matrices, the scaling factor ω may be chosen to be the inverse of the best approximation of the highest eigenvalue of the matrix A $\left(\omega = \dfrac{1}{\lambda_{max}}\right)$.

Observe that this preconditioner does not require the storage of an additional matrix apart from the initial coefficient matrix, but uses a sequence of matrix-vector products. However, the greater the number of terms in (4.94), the better the preconditioner but the more expensive it is to apply. Below, in Algorithm 4.2 we present a Neumann preconditioning algorithm that returns the product between the preconditioner and the residual in a Preconditioned Conjugate Gradient method.

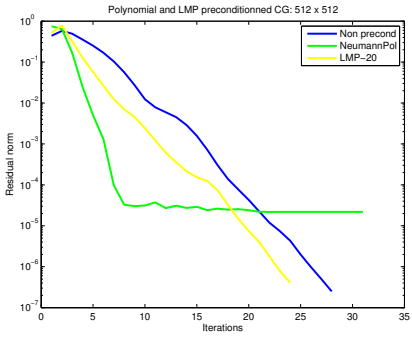
3. The real $\rho(G)$ is the radius of the smallest closed ball centered in zero and containing all the eigenvalues of G. In particular, $\rho(G) = \lambda_{max}$ for G symmetric positive definite.

Algorithm 4.2 ($P_m(A)r$.)

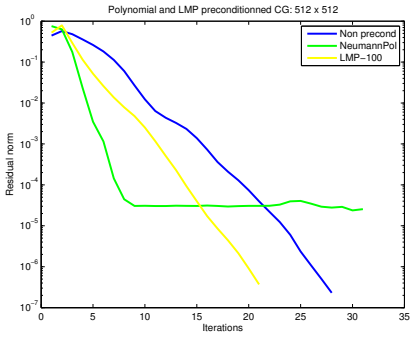
Input :	A,	% *system matrix,*
	I	% *identity matrix,*
	r,	% *vector to be multiplied by* $P_m(A)$,
	$maxIter$	% *maximal number of iterations*
	$\Theta = [a, b]$	% $\approx [\lambda_{min},\ \lambda_{max}]$.
Output:	$z = P_m(A)r$	
	$\omega = \frac{1}{b}$;	
	$z = (I - \omega A)r$;	
For	$(k = 1, 2, \ldots, maxIter)$;	
	$z = (I - \omega A)(r + z)$;	
End(For)		
$\mathbf{z} = \omega(\mathbf{r} + \mathbf{z})$		

Since the number of terms of the preconditioner infers the number of matrix-vector products, it may not be interesting to use the Neumann preconditioner. This can be illustrated in the example below.

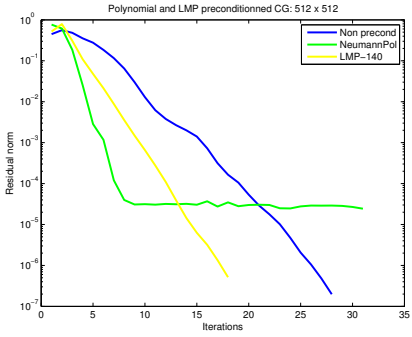
Figure 4.6 compares the LMP and the Neumann polynomial preconditioners to nonpreconditioned CG for the linear system of the form (4.9) of size 512. The number of the Neumann polynomial terms is fixed at 20 and the number of vectors to build the LMP preconditioner is respectively $20, 100, 140$ and 512. We increased significantly the number of vectors in the LMP ($k + 1 = 20, 100, 140, 512$ respectivelly for plots on top left, top right, bottom left and bottom right), while the number of CG iterations decreases slowly ($25, 22, 18$ and 1 respectively). It is observed that in all the cases, none of these preconditioners is efficient since the nonpreconditioned CG can reach the solution with about 30 iterations. The use of the Neumann preconditioner necessitates 20 matrix-vector products times the 8 iterations of the PCG. To sum up, one has done 160 matrix-vector products compared to only 30 in a nonpreconditioned case. Ofcourse some inner products have been saved, but they can not compensate for these additional matrix-vector products. For LMP, things are slightly different. The application of an LMP with $k+1$ vectors (this is the product with the residual vector of length N) costs $10(k+1)N$ operations. This cost is less if k is too small compared to N. However, one observes that for $k + 1 = 20$, the gain in the CG is insignificant (4.6a). The augmentation of the number of vectors in the LMP increases the cost $10(k + 1)N$ of the matrix-vector product exponentially, while the decrease in the number of CG iterations is sublinear ($k + 1 = 20, 100, 140, 512$ respectively for plots (4.6a), (4.6b), (4.6c) and (4.6d)). This means, the gain in decrease of CG iterations to reach the same precision is not sufficient compared to the additional cost due to application of the preconditioner (20×8 matrix-vector for Neumann and $10(k + 1)N$ operations in each LMP iteration.



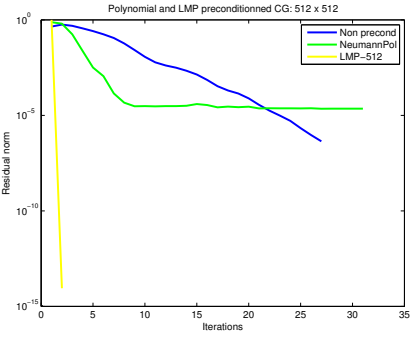
(a) Comparison of LMP and Neumann polynomial preconditioners ($k+1=20$, $m=20$).



(b) Comparison of LMP and Neumann polynomial preconditioners ($k+1=100$, $m=20$).



(c) Comparison of LMP and Neumann polynomial preconditioners ($k+1=140$, $m=20$).



(d) Comparison of LMP and Neumann polynomial preconditioners ($k+1=512$, $m=20$).

Figure 4.6: Comparison of LMP and Neumann polynomial preconditioners.

4.4.3 Tchebychev polynomial preconditioner

Let us consider that $P_m(\lambda)$ is the solution of the problem (see also (4.82))

$$\min_{P_m \in \mathcal{P}_m[\Theta]} \|1 - \lambda P_m(\lambda)\|, \tag{4.95}$$

where $\lambda \in \Theta$ and $\mathcal{P}_m[\Theta]$ is the set of polynomials of a degree not greater than m on the continuous interval Θ containing the spectrum $\sigma(A)$. To achieve an optimal solution of the problem (4.95), one may solve an approximation problem using the norm defined in Annexe C (equation C.4) from the continuous scalar product (4.44). This leads to solve a weighted least-squares problem

$$\min_{P_m \in \mathcal{P}_m[\Theta]} \|1 - \lambda P_m(\lambda)\|_w, \tag{4.96}$$

where $\lambda \in \Theta$ and whose solution is the sought polynomial preconditioner. When the 2-norm is used in (4.82), the preconditioner is said to be a least-square polynomial preconditioner.

The Tchebychev polynomial preconditioner is the one that chooses to minimize the uniform norm by solving the problem

$$\min_{P_m \in \mathcal{P}_m} \max_{\lambda \in \Theta} |1 - \lambda P_m(\lambda)|. \tag{4.97}$$

Setting $\varepsilon_{m+1} = \min_{P_m \in \mathcal{P}_m} \max_{\lambda \in \Theta} |1 - \lambda P_m(\lambda)|$, it can be shown that (see [69, p.59])

$$k(P_m(A)A) \leq \frac{1 + \varepsilon_{m+1}}{1 - \varepsilon_{m+1}}. \tag{4.98}$$

This ensures that the condition number is reduced whenever $\varepsilon_{m+1} < 1$. The linear map $\lambda \to \Psi(\lambda) = \frac{b+a-2\lambda}{b-a}$ is required to map the interval $\Theta = [a, b]$ onto the interval $[-1, 1]$ where the polynomial $T_m(\lambda)$ (see (4.39)) has minimal uniform norm. taking $\phi_{m+1}(\lambda) = 1 - \lambda p_m(\lambda)$, one can verify that the solution of (4.97) [69] is characterized by the following result.

Proposition 4.10

Let $\lambda \in \Theta = [a, b] \subset \mathbb{R}$, where $0 < a < b$. Then

$$\min_{p_m \in \mathcal{P}_m} \max_{\lambda \in \Theta} |1 - \lambda p_m(\lambda)| = \left| \frac{T_m\left(\Psi(\lambda)\right)}{T_m\left(\Psi(0)\right)} \right| = \left| \frac{T_m\left(\frac{b+a-2\lambda}{b-a}\right)}{T_m\left(\frac{b+a}{b-a}\right)} \right| = \hat{p}_m(\lambda), \tag{4.99}$$

where $T_m(\lambda)$ is the Tchebychev polynomial of first kind and degree m ((4.39)).

Assuming

$$\theta = \frac{b+a}{2}, \quad \delta = \frac{b-a}{2}, \quad \sigma_k = T_k\left(\frac{\theta}{\delta}\right), \tag{4.100}$$

and using the previous result (4.99), one can verify that

$$\hat{p}_k(\lambda) = \frac{1}{\sigma_k} T_k\left(\frac{\theta - \lambda}{\delta}\right). \tag{4.101}$$

Linking all this, we have the relations explained in [76, pp.196-198].

$$\begin{aligned}
&\sigma_0 = 1,\\
&\sigma_1 = \frac{\theta}{\delta},\\
&\sigma_{k+1} = 2\frac{\theta}{\delta}\sigma_k - \sigma_{k-1},\\
&\rho_0 = \frac{1}{\sigma_1},\\
&\rho_k = \frac{1}{2\sigma_1 - \rho_{k-1}},\ k \geq 1,\\
&\phi_0(\lambda) = 1,\\
&\phi_1(\lambda) = 1 - \frac{\lambda}{\delta},\\
&\phi_{k+1}(\lambda) = \rho_k[2(\sigma_1 - \frac{\lambda}{\delta})\phi_k(\lambda) - \rho_{k-1}\phi_{k-1}(\lambda)],\ k \geq 1.
\end{aligned}$$

This leads to the algorithm (4.3) of the Tchebyshev preconditioner [68, p.383]

Algorithm 4.3 (*Tchebychev preconditioner*)

Input :	A	$\%$ *system matrix*;
	r;	%vector to be applied by$P_m(A)$;
	$maxIter$;	$\%$ *max number of iterations*
	$\Theta = [a, b]$;	$\% \approx [\lambda_{min}, \lambda_{max}]$;
Output:	$z = P_m(A)r$;	
Initialization		
	$\theta = \frac{b+a}{2}$; $\delta = \frac{b-a}{2}$; $\sigma_0 = 1$; $\sigma_1 = \frac{\theta}{\delta}$;	
	$\rho_0 \leftarrow \frac{1}{\sigma_1}$; $z_0 = 0$; $r_0 = r$; $d_0 = \frac{1}{\theta}r$;	
	$\%$ *Loop*	
For	$(k = 1, 2, \ldots, maxIter)$;	
	$z_k = z_{k-1} + d_{k-1}$	
	$r_k = r_{k-1} + Ad_{k-1}$	
	$\rho_k = \frac{1}{2\sigma_1 - \rho_{k-1}}$;	
	$d_k = \rho_k \rho_{k-1} d_{k-1} + \frac{2\rho_k}{\delta} r_k$;	
End(For)		
	$z = z_k$	

Note that, there is no inner product in this algorithm. The cost of this preconditioner is mainly the three vector updates in addition to the matrix-vector product. Furthermore, as we stated in the motivation, in opposition to stationary preconditioners (Ichol, see Subsections 4.2.3 and Jacobi, GS, SOR, see Subsection 4.2.4) the user can control the degree of the polynomial. This is a common point with the LMP preconditioner (see Sections 4.3) and Neumman polynomial preconditioner (see Subsection 4.4.2). However, to fix the degree of the polynomial is a main issue because it impacts the number of matrix-vector products. This may be relatively large for sufficient decrease in the min-max criteria (4.99) and, in general, the best degree is not known in advance. A compromise is needed to fix the degree (low degree implies low cost but also less improvement on the CG convergence rate and high degree implies higher cost but higher improvement on the CG convergence rate. We will see in Section 4.6 how the cost of this precoditioner impacts the total cost of the PCG algorithm.

4.5 Systems from Gauss-Newton-like algorithm

The system from Gauss-Newton like algorithm is given by the equation (2.50) and its derivation is explained in subsection (1.4.1). It has the general form

$$Ax = b, \quad \text{with} \quad A = D^T D + J^T J \tag{4.102}$$

where D is a discrete differential operator and J is the Jacobian of a residual function (2.46). In this thesis, The considered unidimensional differential operator is given by (1.33). One can verify that, ${D^{(k)}}^T D^{(k)} = \Delta^{(k)}$ and the equation (4.102) writes simply

$$A = D^T D + J^T J, \tag{4.103}$$

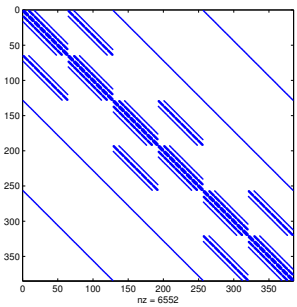
where $D^T D$ may be A_{el} or A_{dif} depending on the choice of D (see Annexe B: equation B.2 or equation B.3) and J is the derivatives of the residual functions, that are

cubic spline functions obtained from interpolation of the images. The derivatives can be computed following (1.38). This second part of the matrix A above is given by

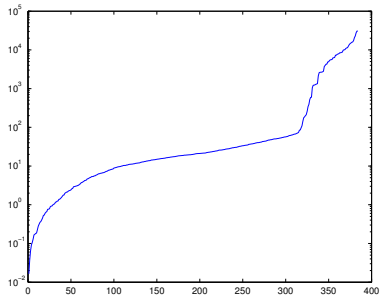
$$J^TJ = \begin{bmatrix} h_{111} & & & h_{112} & & & h_{113} & & \\ & \ddots & & & \ddots & & & \ddots & \\ & & h_{N11} & & & h_{N12} & & & h_{N13} \\ h_{112} & & & h_{122} & & & h_{123} & & \\ & \ddots & & & \ddots & & & \ddots & \\ & & h_{N12} & & & h_{N22} & & & h_{N23} \\ h_{113} & & & h_{123} & & & h_{133} & & \\ & \ddots & & & \ddots & & & \ddots & \\ & & h_{N13} & & & h_{N23} & & & h_{N33} \end{bmatrix} \in \mathbb{R}^{dN \times dN} \tag{4.104}$$

where $h_{ikl} = (\nabla I_m[\Phi(X_i^{(k)}])(\nabla I_m[\Phi(X_i^{(l)})),\ i = 1 : N,\ k, l = 1 : d$ and I_m is the moving image and ∇I_m represents the derivatives of the moving image.

Figure 4.7 below visualizes the sparsity patterns of A Figure 4.7a while its spectrum is visualized by Figure 4.7b for image of size $[448],\ N = 384$.



(a) Sparsity patterns of the matrix A in (4.102)



(b) Sorted spectrum of H $n^{(1)} = 4, n^{(2)} = 4, n^{(3)} = 8 : N = 384$

Figure 4.7: Sparsity and spectrum structures of the matrix $A = D^TD + J^TJ$ in (4.102)

The matrix A is obviously symmetric and positive definite. The symmetry is obtained by construction. It is positive definite thanks to the first term operator (see Section 4.1). Indeed, although the J^TJ matrix is symmetric, it is positive definite only when it has full rank. This is not always the case, in particular for Jacobian from image interpolation. This also means that, it is likely that A has worse condition number than A_{el} and A_{dif} since the smallest eigenvalue is often zero in J^TJ while the highest will increase in all cases (unless the Jacobian is zero).

Preconditioning A is more complicated since the coefficient matrix is changing after each inner loop and analytic extreme eigenvalues are not known. However, since the structure of the first term dominates the second term (observe that J^TJ has ony

5 diagonals and the 2 sub diagonals are equal to the 2 super diagonals), thus we may apply the same conditioning techniques.

In the following section, we compare splitting preconditioners (Jacobi and Symmetric Gauss-Seidel 4.2.4) with Tchebychev polynomial preconditioner 4.4.3, Neumann polynomial preconditioner 4.4.2 and Incomplete Cholesky preconditioner 4.2.3 on images described in Chapter 3 within the local computer described there in.

4.6 Numerical experiments for system solvers

In this section we present numerical experiments for various preconditioners in the CG algorithm 2.2 The goal is to compare the effectiness of polynomial preconditioners (Neumann and Tchebychev) compared to splitting preconditioner (Jacobi and SGS) and the default icomplete Cholesky. The PCG algorithm was applied to solve a linear system of the form (4.9) from the 3D brain image registration within each of these preconditioners. The image brain is one of those presented in Chapter 3 and the tests are done on 4 levels (see 3.1) of the image to observe the sensitivity of each algorithm to the size of the system and thus the step size of the discretization. The matrix is thus a 3D Laplacian where the size is denoted N and varies for each level. $nnz(A)$ denotes the number of non zeros elements in the matrix A.

The stopping criteria is considered as the convergence test on the PCG solvers. Since we aim to solve the problem

$$\min_{x\in\mathbb{R}^N} \|b - Ax\|_2$$

by setting

$$r_k = b - Ax_k$$

where x_k is the solution computed at iteration k, we use the stopping criteria (see [49])

$$\|r_k\|_2 < \delta_1 \quad \text{or} \quad \frac{\|A^T r_k\|_2}{\|r_k\|_2} \leq \frac{\|A^T r_0\|_2}{\|r_0\|_2} * \delta_2 \tag{4.105}$$

where $\delta_1 = 10^{-6}$ and $\delta_2 = 10^{-5}$. As stated by Gould and Scott [49], this stopping criteria allows to compare different preconditioners since it does not depend on the used preconditioner. In addition, this stopping criteria is appropriate for our problems since our systems are normal equations as it was stated in Section 2.3. The maximum number of iterations was set to 500.

Practical considerations to compute the cost of each preconditioner is presented below. Let be given a matrix A of size N and with $nnz(A)$ number of nonzeros elements in A. We consider each fundamental operation (addition, multiplication, substraction, multiplication) as a single `flop`. Hence, we approximate the costs of the matrix-vector product (matvec) by $C_{mv} \approx 2nnz(A)$, the dot product of two vectors (inner product) by $C_{dot} \approx 2N$ and the vector update by $C_{axpy} \approx 2N$. This allows us to approximate the cost of each preconditioner. Note that, the Jacobi preconditioner costs almost

$$C_{Jac} \approx 2N$$

since this requires simple multipication with the inverse of the diagonal. The cost of the Gauss-Seidel preconditioner and the cost of The Incomplet Cholesky factorization with no fill-in can be approximated (see [68]) by

$$C_{SGS} \approx 3N + 2nnz(A)$$

and

$$C_{Ic(0)} \approx 3N + 2nnz(A).$$

For non stationnary preconditioners, it can be noticed that in Neumann preconditioner one requires one matvec product and two updates at each iteration. For Tchebychev polynomial, one needs one matvec product and three updates. Thus, for a Neumann polynomial of degree m we have

$$C^m_{Neum} \approx m(C_{mv} + 2C_{axpy}),$$

while

$$C^m_{Tcheb} \approx m(C_{mv} + 3C_{axpy}),$$

for a polynomial of degree m. Converting this in term of N and $nnz(A)$ gives the Table 4.2.

Operation	**Notation and Cost(flops)**
Jacobi	$C_{Jac} \approx 2N$
SGS	$C_{SGS} \approx 3N + 2nnz(A)$
Ic(0)	$C_{Ic(0)} \approx 3N + 2nnz(A)$
Neumann	$C_{Neum} \approx m(2nnz(A) + 4N)$
Tchebychev	$C_{Tcheb} \approx m(2nnz(A) + 6N)$

Table 4.2: Flops count for preconditioners. N is the size of the matrix system, $nnz(A)$ is the number of nonzeros entries in A and m is the degree of polynomial.

Since we know that the CG requires per iteration one matrix-vector product, three updates and two dot products, the cost of one CG iteration is

$$C_{cgIt} = C_{mv} + 3C_{axpy} + 2C_{dot} = 2nnz(A) + 10N.$$

Thus, to estimate the PCG cost of each solver we have

$$\begin{array}{lcl} C_{pcgJac} & \approx & Itcg \times (C_{cgIt} + C_{JAC}), \\ C_{pcgSGS} & \approx & Itcg \times (C_{cgIt} + C_{SGS}), \\ C_{pcgIc(0)} & \approx & Itcg \times (C_{cgIt} + C_{Ic(0)}), \\ C_{pcgNeum} & \approx & Itcg \times (C_{cgIt} + C_{Neum}), \\ C_{pcgTcheb} & \approx & Itcg \times (C_{cgIt} + C_{Tcheb}). \end{array} \tag{4.106}$$

4.6.1 Comparison of the PCG convergence with different preconditioners on the braine and Chest images

In this subsection, we pick up two images to compare in details five preconditioners on these specific images. A more inclusive comparison is done in the next subsection using the performance profile tool. The only reason for what the choice of these two images was done is the density (ratio between the number of nonzero entries and the length of the vector) of the right-hand side b at each level. In fact, the density of the brain image was, in average, the highest $\approx 99,9\%$ while the density of the Chest image was among the lowest $\approx 68,9\%$. Unfortunately, this did not lead to particular conclusion. The test was done on my computer described in Chapter 3.4.

Table 4.3 and Table 4.4 present, respectively on the Brain and the Chest images, a comparison of the Jacobi preconditioner (Jacobi), the Symmetric Gauss-Seidel preconditioner (SGS), the default Incomplete Cholesky (Ichol) this means with fill in, the Neumann polynomial with degree $m = 50$ and the Tchebychev polynomial with degree $m = 50$. The operations counts are expressed in flops and $1\texttt{Mflops} = 10^6$ flops. The 1^{st} column presents the preconditioner (Precond), the 2^{nd} is the polynomial degree (for Neumann and Tchebychev), the third column describes the number of CG iterations, the 4^{th} column presents the cost of the preconditioner in terms of Millions of flops based on Table 4.2, ($\mathbf{C_{prec}} = C_{pcgJac}$ for Jacobi preconditioner, $\mathbf{C_{prec}} = C_{pcgSGS}$ for SGS preconditioner and so on), the 5^{th} column presents the cost of each CG iteration, the 6^{th} column presents the total cost of the PCG algorithm with the given preconditioner based on (4.106). The norm of the residual (4.105) is presented in the 7^{th} column while the computing time in seconds, computed using the tic-toc `Matlab` command, is presented in the 8^{th} column. Values with an $*$ denote that the pcg did not converge. One can eliminate the Jacobi preconditioner (since it does not converge for certain systems), eliminate the Cholesky preconditioner (since it does not adress the large systems) and Neumann preconditioner since it is the most expensive both in computing time and in flops count. It remains to compare SGS and Tchebychev preconditioners. From this table, it is clear that the Tchebychev polynomial preconditioner is the winner for large systems while SGS may be prefered for less large systems in terms of flops count. The most important observation concerns the computing time. One can observe that, even when the Tchebychev preconditioner is expensive in terms of flops count, it converges faster. This may be explained by its natural and easy parallelization. This confirms the assumption stated in the motivation of Section 4.4. From this table, one can verify that, for small images (level 4 and 5). The Incomplete cholesky is the winner in term of computing time but it is the most expensive in term of flops. Thus, definitly, SGS and Tchebychev are to be preferred.

Figure 4.8 illustrates the results of Table 4.3 while Figure 4.9 illustrates the results of Table 4.4

To benchmark these PCG solvers, now all the database of our images is used and we use a profile performance to benschmark them. This is presented in the following subsection.

4.6.2 Performance profiles of system solvers

In this subsection, we are especially interested in benchmarking the iterative system solvers, presented in the previous subsection The bechmark is done on the large set of 3D medical images presented in Chapter 3. Although it is recognized (see for example [49]) that it is rare to get a single system solver that is the best for all the problems, results of performance profile indicate which solver has the highest probability of being the best within a factor $f \in [1, \infty[$ (see 3.2, here f has the role of α) and considering a limited computional budget in terms of time and memory requirements.

In this work, we follow the methodology in [49] to compare PCG solvers described in 4. The set of solvers contains 5 PCG solvers, that is

$$n_s = \#\mathcal{S} = 5.$$

These solvers are run on the set of 54 problems formed by 3D images from Ta-

precond	**m**	**Itcg**	$\mathbf{C_{prec}}$ (*Mflops*)	$\mathbf{C_{cgIt}}$ (*Mflops*)	$\mathbf{C_{pcg}}$ (*Mflops*)	**residual norm**	**time(s)**
Level 4, $\mathbf{N} = 6144$, $\mathbf{nnz}(\mathbf{A}) = 132576$							
Jacobi	–	500	1.2×10^{-2}	0.32	1.66×10^{2}	$4.8\times10^{-6*}$	0.13
SGS	–	78	4.5×10^{-2}	0.32	2.84×10^{1}	9.5×10^{-7}	0.15
Ichol	–	1	7.7×10^{4}	0.32	7.7×10^{4}	3.2×10^{-7}	0.23
Neumann	50	78	2.5×10^{0}	0.32	2.19×10^{2}	9.2×10^{-7}	1.10
Tchebychev	50	4	3.2×10^{0}	0.32	1.40×10^{1}	4.9×10^{-7}	0.19
Level 5, $\mathbf{N} = 49152$, $\mathbf{nnz}(\mathbf{A}) = 1\,143\,744$							
Jacobi	–	500	9.8×10^{-2}	2.7	1.39×10^{3}	$3.5\times10^{-6*}$	1.35
SGS	–	119	2.4×10^{0}	2.7	6.10×10^{2}	9.0×10^{-7}	2.79
Ichol	–	5	3.9×10^{7}	2.7	1.95×10^{8}	1.4×10^{-7}	17.72
Neumann	50	82	1.2×10^{2}	2.7	1.03×10^{4}	9.1×10^{-7}	15.41
Tchebychev	50	4	1.2×10^{2}	2.7	5.26×10^{2}	9.2×10^{-8}	2.25
Level 6, $\mathbf{N} = 393216$, $\mathbf{nnz}(\mathbf{A}) = 9\,488\,256$							
Jacobi	–	500	7.0×10^{-1}	22.9	1.18×10^{4}	$1.9\times10^{-5*}$	19.31
SGS	–	172	2.0×10^{1}	22.9	7.37×10^{3}	9.2×10^{-7}	36.24
Ichol	–	–	–	–	–	–	
Neumann	50	118	1.0×10^{3}	22.9	1.23×10^{5}	4.2×10^{-7}	321.34
Tchebychev	50	6	1.1×10^{3}	22.9	6.49×10^{3}	1.6×10^{-8}	45.73
Level 7, $\mathbf{N} = 3\,145\,728$, $\mathbf{nnz}(\mathbf{A}) = 77\,270\,784$							
Jacobi	–	500	6.3×10^{0}	185	9.56×10^{4}	$9\times10^{-4*}$	49.9
SGS	–	244	1.6×10^{2}	185	8.41×10^{4}	9.7×10^{-7}	223
Ichol	–	–	–	–	–		
Neumann	50	167	8.3×10^{3}	185	1.41×10^{6}	9.8×10^{-7}	1180
Tchebychev	50	8	8.6×10^{3}	185	7.02×10^{4}	9.6×10^{-7}	145.6

Table 4.3: Comparison of the PCG convergence with different preconditioners on the brain image, level 4 to level 7

ble 3.1 where each level of an image is considered as a different problem. Thus

$$n_p = \#\mathcal{S} = 54.$$

The stopping criteria (4.105) is used here as the convergence test and the maximal number of iterations is set to 500. Hence, a problem is considered as solved if the convergence test is verified within 500 iterations.

Figure 4.10 presents a comparison of performance profiles for PCG solvers with respect to the number of iterations. Consider at one hand, $f = 1$, comparing directly the performance ratios of the five solvers. One can see that the PCG with Tchebychev polynomial preconditioner is the winner over all the others and the Jacobi preconditioner is the worse. The SGS preconditioner is competitive with the Neumann polynomial but one can observe from Table 4.3 and Table 4.4 that the Neumann preconditioner is the most expensive. Consider at the other hand, $f = 40$, and observe that, respectively, $\rho_{SGS}(40) \approx 94\%$, $\rho_{Ichol}(40) \approx 81\%$, $\rho_{Jacobi}(40) \approx 5\%$,

precond	**m**	**Itcg**	$\mathbf{C_{prec}}$ (*Mflops*)	$\mathbf{C_{cgIt}}$ (*Mflops*)	$\mathbf{C_{pcg}}$ (*Mflops*)	**residual norm**	**time(s)**
Level 4, $\mathbf{N} = 3072$, $\mathbf{nnz(A)} = 63744$							
Jacobi	–	249	6.1×10^{-2}	0.15	5.25×10^{1}	8.3×10^{-7}	0.11
SGS	–	72	1.3×10^{-1}	0.15	2.01×10^{1}	9.8×10^{-7}	0.11
Ichol	–	2	9.6×10^{3}	0.15	1.92×10^{4}	9.3×10^{-7}	0.09
Neumann	50	50	6.9×10^{0}	0.15	3.52×10^{2}	9.9×10^{-7}	0.54
Tchebychev	50	4	7.3×10^{0}	0.15	2.98×10^{1}	4.1×10^{-7}	0.21
Level 5, $\mathbf{N} = 24576$, $\mathbf{nnz(A)} = 561\,408$							
Jacobi	–	397	4.9×10^{-2}	1.36	19.5	9.9×10^{-7}	1.003
SGS	–	110	1.2×10^{0}	1.36	2.80×10^{2}	9.6×10^{-7}	0.99
Ichol	–	4	4.9×10^{6}	1.36	1.96×10^{7}	1.6×10^{-8}	3.29
Neumann	50	76	6.1×10^{1}	1.36	4.73×10^{3}	8.5×10^{-7}	7.68
Tchebychev	50	4	6.3×10^{1}	1.36	2.57×10^{2}	5.0×10^{-7}	1.4
Level 6, $\mathbf{N} = 196608$, $\mathbf{nnz(A)} = 471\,696$							
Jacobi	–	500	3.9×10^{-1}	2.9	1.64×10^{3}	8.3×10^{-6}*	6.86
SGS	–	161	1.5×10^{0}	2.9	7.13×10^{2}	9.9×10^{-7}	18.5
Ichol	–	–	–	–	–		
Neumann	50	111	8.7×10^{1}	2.9	9.92×10^{3}	9.1×10^{-7}	110
Tchebychev	50	6	1.1×10^{2}	2.9	6.53×10^{2}	1.6×10^{-7}	14.71
Level 7, $\mathbf{N} = 3\,145\,728$, $\mathbf{nnz(A)} = 77\,270\,784$							
Jacobi	–	500	3.1×10^{0}	185	9.40×10^{4}	5.5×10^{-4}*	56.5
SGS	–	233	8.1×10^{1}	185	6.19×10^{4}	9.5×10^{-7}	254.3
Ichol	–	–	–	–	–		
Neumann	50	159	4.2×10^{3}	185	6.60×10^{5}	9.4×10^{-7}	1265
Tchebychev	50	8	4.3×10^{3}	185	3.59×10^{4}	4.1×10^{-7}	208

Table 4.4: Comparison of the PCG convergence with different preconditioners on the chest image, from level 4 to level 7.

$\rho_{Cheby}(40) \approx 100\%$, $\rho_{Neumann}(40) \approx 94\%$. This means that the SGS, the Ichol, the Jacobi and the Neumann solvers, respectively, require up to 40 times the efforts (iterations) of the best solver (Cheby) to solve, respectively, almost 94%, 81%, 5%, 94% of problems.

As a conclusion to this chapter, we suggest the use of the Tchebychev polynomial preconditioner or SGS preconditioners to address linear systems from 3D medical image registration.

However, very large images and when the induced system are highly structured as (4.9), one may think of using more appropriate techniques. Such techniques include numerical tensor methods that we present in the following chapter.

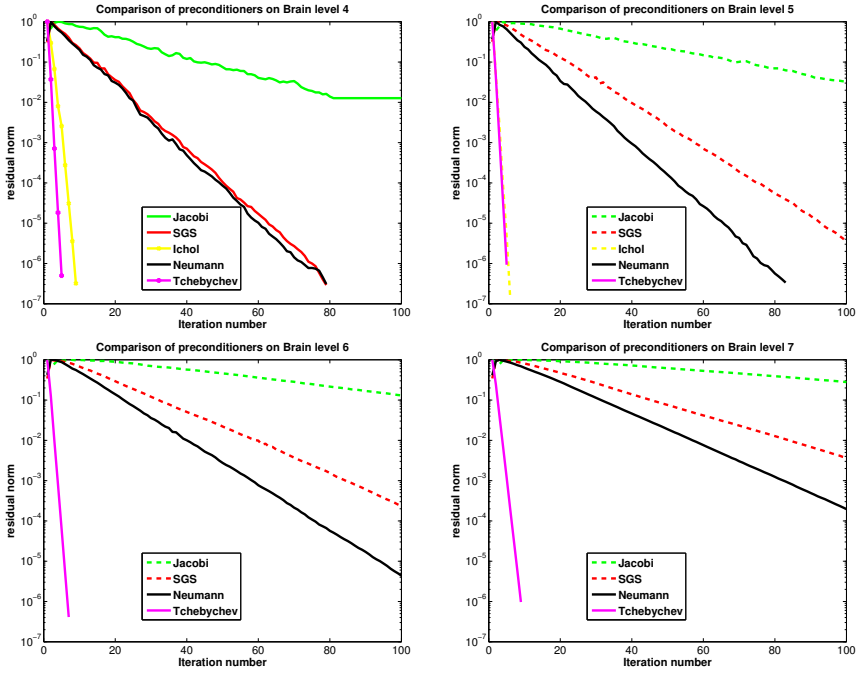


Figure 4.8: Comparison of the PCG convergence with different preconditioners on linear systems from 3D brain registration

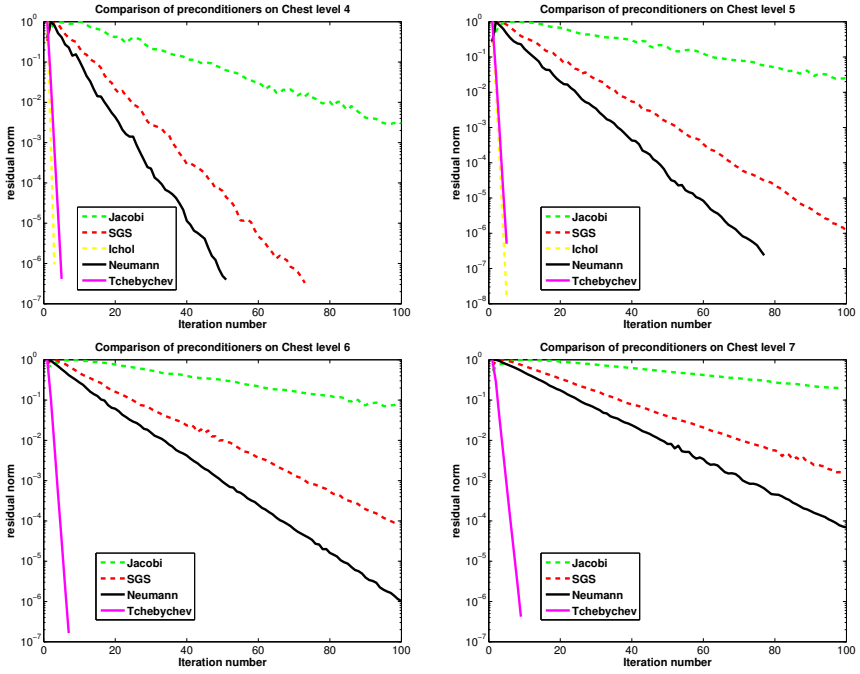


Figure 4.9: Comparison of the PCG convergence with different preconditioners on linear system from 3D chest registration.

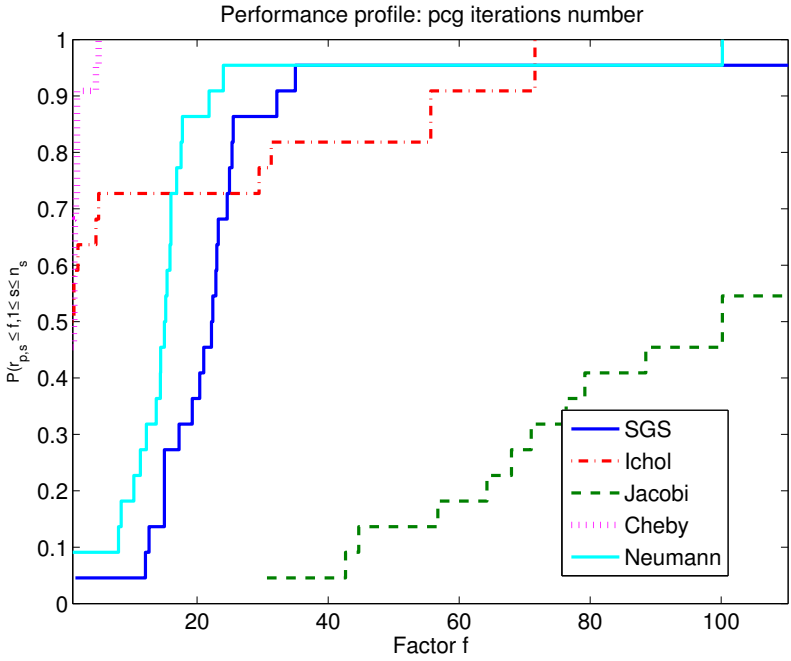


Figure 4.10: Comparison of performance profiles of PCG solvers with respect to the number of iterations on 54 registration problems (3D images).

Chapter 5

Numerical tensor formats and linear systems in 3D MIRP

Introduction

As stated in previous chapters, the medical image registration problem needs to solve a sequence of linear systems during the optimization process. Although these systems are often sparse and structured, they are very large and ill-conditioned. Thus, their system solvers are time consuming and their complexity in operation counts is polynomial. In addition, although in best cases (diagonal matrices) they can achieve linear complexity, this linear complexity may be unsatisfactory for very large and high-dimensional images.

In Chapter 1, we have presented the Demons algorithms (this does not solve explitly the linear systems (see 1.4.2)) whose complexity is linear $\mathcal{O}(N)$ (where N is the number of image voxels). In Chapter 4, we have presented certain algorithms with quasi linear complexity $\mathcal{O}(NlogN)$ based essentially on Fast Discrete Sine Transforms (FDST), Fast Fourier Trabsforms (FFT) or Additive Splitting Operators (ASO). However, these linear and quasi linear complexities are obtained by direct methods known to scale poorly in terms of operation counts and memory consuming for very large problems arising from PDEs discretization [55].

Conversely, iterative methods may require fewer operations and storage when an approximation of a solution with relatively low accuracy is needed [55]. Although these methods are considered reliable compared to direct methods for high-dimensional and large problems, they may fail or be unable to achieve the desired accuracy. Especially for high dimensional PDEs problems, iterative methods require reliable preconditioners. The construction and application costs of the preconditioner may be higher, making the iterative solver less attractive.

Another way of accelerating the registration process is the development of algorithms which offer the best compromise between complexity and speed [1]. Over the last few years, compression techniques for high-dimensional data using suitable data sparse representations have been developed [6]. In general, these representations

known as *numerical tensor representations* are inspired by variables separability, hierarchical matrix techniques and low-rank approximations (see for example [6, 7, 8] and references therein). In this chapter, we use numerical tensor techniques to solve certain large linear systems from MIRP.

A numerical tensor can be defined informally as a multidimensional array where the position of each entry is determined by multiple indices. The number d of indices used to determine the position of an entry is called the *order* or *dimension* of the tensor. So, a tensor of order d, has d principal directions (axes) that will be called generally *modes* of the tensor.

With respect to large linear systems, tensor representations allow the transformation of these linear systems into appropriate multilinear systems and exploit truncation, sparsity and parallelization of the process to reduce the computational cost and time. In addition, calculations in each unidimensional subspace of the problem may be enabled.

Note that, numerical tensors have been used for decades in data processing and analysis of multidimensional phenomena. In general, numerical tensor elements and operations are considered as generalizing vectors, matrices and operations from linear algebra (dimension $d = \{1, 2\}$) in a multidimensional context (dimension $d > 2$) to be explored by multilinear algebra. However, although tensor spaces are generated from vector spaces, there are certain concepts from vector spaces whose properties are not straightforward generalized in tensor spaces of dimension greater than two [8, p.15].

For example, the best low-rank approximation (in the sense of least-squares) for matrices ($d = 2$) may be provided by the singular value decomposition (SVD) via the well known Eckhardt-Young theorem (mentioned for example in [77]). Unfortunately, the concept of *rank* that is uniquely defined for matrices, is not uniquely defined for general tensors with $d > 2$. Hence, the theorem mentioned above is not straightforward generalized in dimension $d > 2$. The idea of MultiLinear Singular Value Decomposition (MLSVD) (see for example [50], [78]) and High Order Singular Value Decomposition (HOSVD) (see for example [6]) came to fill in this gap.

However, it has been shown (see [6]) that under certain conditions, some tensor representations may enable to reduce the complexity, in term of number of parameters, of matrix-data of size $[n^d \times n^d]$, $n, d \in \mathbb{N}$ from $\mathcal{O}(n^d)$ to $\mathcal{O}(dlog(n))$ in a d-dimensional space. It is then important to transform such matrices and vectors in tensor formats expecting that the complexity will be reduced significantly by compressing and truncating the data.

For medical image registration, numerical tensors have been used for example in [79], where the authors approximated a nonseparable filter by a low-rank separable filter based on tensor approximation. A rank-one tensor via Tucker decomposition was approximated and used to replace a large matrix filter. The authors were able to obtain a significant gain in computational time and storage compared to classical methods (without numerical tensor tools). However, the algorithm that the authors proposed could not preserve the precision of the classical methods, since the rank-one approximation was not enough for a good filter approximation.

In this thesis, We aim to speed up the registration process with tensor techniques by offering possibilities to replace a linear complexity $\mathcal{O}(N)$ by a quasi logarithmic complexity $\mathcal{O}(d \log N)$, where $N \in \mathbb{N}$ is the number of image voxels and d is the image dimension. However we focus only on the system solver, which is often the most expensive step in the process. This supposes the use of a compressed representation of data with a given accuracy ε ($0 \leq \varepsilon < 1$). Certainly, this needs a good understanding

of both the acquisition process of the images and the application at hand. For this purpose, we propose an adapted Preconditioned Conjugate Gradient algorithm [40] to solve the large linear systems

$$MA\mathbf{x} = M\mathbf{b}, \quad M, A \in \mathbb{R}^{N\times N},\ \mathbf{x}, \mathbf{b} \in \mathbb{R}^N, \tag{5.1}$$

in appropriate tensor format. The system (5.1) arises from $3D$ deformable medical image registration and has to be reformulated in a particular tensor representation (here the Tensor-Train representation 5.3).

On the one hand, the matrix A is a three-dimensional Laplace-like operator, for which it is possible to reorganize indices of components as products of new multi-indices that admits variables separability [80, 81]. For this purpose, the operator A is expressed as a sum of Kronecker products and the preconditioner M is a low-rank approximation of its inverse, computed via eigen elements. On the other hand, the right-hand side $\mathbf{b}$ contains information from the images. The success of its low-rank approximation depends on the nature (modality) of the images and this infers the quality of the low-rank linear system solution $\mathbf{x}$ as this will be illustrated numerically in Chapter 6.

In this chapter, we recall the main ingredients for processing numerical tensors. The main definitions and operations are addressed in Section 5.1 while the commonly used formats are presented in Section 5.2. The focus is on Tensor-Train format in Section 5.3 before using these formats in Section **??** to construct a low-rank preconditioner for multidimensional Laplacian operators for linear systems arising in medical image registration.

5.1 General techniques with numerical tensors

This section concerns general concepts and operations that are useful for a good understanding of system solvers in numerical tensor formats. In general, the number of entries in a numerical tensor increases exponentially with respect to the number of dimensions of the tensor. For multilinear systems in particular, the number of unknowns and the number of equations have an exponential rate of increase with respect to the dimension. This exponential rate of increase is commonly referred to as the *curse of dimenionality* (see [82]) and is reflected into the computational and the storage requirements. In scientific computing, certain tensor representations and tensor decompositions, such as Canonical Polyadic (CP), Tensor-Train (TT) and Hierarchical Tensor (HT), have been developed to cope with this curse of dimensionality (see [6], [82]). However, the best approximation is not guaranteed to exist for the CP representation for $d > 3$ and tensor rank greater than 1. In addition, numerical instabilities are reported in the CP representation (see [83], [84]). Thus, the focus will be on TT representations.

To illustrate the importance of referring to tensor methods for the application in this thesis, consider a registration problem of two 3D images ($d = 3$,) whose equal size is $[n^{(1)}, n^{(2)}, n^{(3)}]$. For simplicity we assume $n^{(1)} = n^{(2)} = n^{(3)} = n$. In this case, The problem induces successive linear systems

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{N\times N},\ \ \mathbf{x}, \mathbf{b} \in \mathbb{R}^N.$$

where one has to treat N equations with N unknowns with $N = n^3$. For example, if $n = 1024$, one needs to solve a linear system of size $N = n^3 \approx 2^{30}$, exceeding

the capacity of many standard computers. However, considering $\mathbf{x}, \mathbf{b}$ as tensors of $\mathbb{R}^n \otimes \mathbb{R}^n \otimes \mathbb{R}^n$, for appropriate A and under suitable conditions, the exponential cost n^3 can be reduced to $\mathcal{O}(3n)$ (see [6, p. 11]). To perform these operations, the matrix A has to be written as a sum of *Kronecker products* and thus be considered as a multilinear map that acts on tensor elements in $\mathbb{R}^n \otimes \mathbb{R}^n \otimes \mathbb{R}^n$.

5.1.1 Notations

In the remainder of this chapter, unless specific stated indications, we will use calligraphic upper case, bold upper case, upper case, bold lower case, and lower case to denote, respectively, a high-order tensor space, high-order tensor, a matrix or a vector space, a vector and a scalar. For example, we may denote a tensor space by $\mathcal{A}$, a tensor by $\mathbf{A}$, a matrix by A, a vector space by V, a vector by $\mathbf{a}$, and a scalar by a.

The notation $V^{(k)}$ denotes a vector space of dimension $n^{(k)}$ and thus

$$V^{(1)}, V^{(2)}, \ldots, V^{(d)}$$

indicate vector spaces generating the tensor space

$$\mathcal{V} = \bigotimes_{k=1}^{d} V^{(k)}.$$

When other d-tuple vector spaces are needed, we will use letters

$$W^{(k)}, X^{(k)}$$

and the involved vector spaces are assumed to be defined over the same field $\mathbb{K}$ (here $\mathbb{K} = \mathbb{R}$). Vectors from $V^{(k)}$ are denoted $\mathbf{v}^{(k)}$ and when there are a family of such vectors we use a further index with subscript k. For example, $\mathbf{v}^{(k)}_{j_k}, j_k = 1, 2, \ldots, n^{(k)}$ may denote $n^{(k)}$ vectors of the vector space $V^{(k)}$.

In this chapter, we will refer to the following index sets:

$$\begin{aligned} D &= \{1, 2, \ldots, d\}, \\ I_1 &= \{1, 2, \ldots, n^{(1)}\}, \\ I_2 &= \{1, 2, \ldots, n^{(2)}\}, \\ &\vdots \\ I_d &= \{1, 2, \ldots, n^{(d)}\}. \end{aligned}$$

Using the index set

$$I_k = \{1, 2, \ldots, n^{(k)}\}$$

with cardinality $\#I_k = n^{(k)}$, the vector $\mathbf{v}^{(k)}$ with components

$$\left(v^{(k)}_1, v^{(k)}_2, \ldots, v^{(k)}_{n^{(k)}}\right)$$

may be denoted by

$$\mathbf{v}^{(k)}_{j_k},\ v^{(k)}_{j_k} \in \mathbb{R},\ j_k \in I_k,\ .$$

A tensor is said to be *symmetric* [85] when its array of indices is invariant under permutations:

$$\mathbf{A}_{j_1,j_2,\ldots,j_d} = \mathbf{A}_{\sigma(j_1,j_2,\ldots,j_d)},$$

where σ is any permutation. This property may not reappear in the remaining, but it is implicitly used to reorganize indices in some tensor representations. Thus, we may write $V^{(k)} = \mathbb{R}^{I_k}$ instead of $V^{(k)} = \mathbb{R}^{n^{(k)}}$, to emphasize that permutations of indices are allowed. This means that, ordering in the sets $I_k, k = 1, 2, \ldots, d$ may be of less importance since indices of the entries are elements of their Cartesian product. For example, of between them such as $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ denotes a d-dimensional tensor whose entries are located by elements of the Cartesian product of $I_k, k = 1, \ldots, d$. However, we may still be denoting

$$\mathbf{A} \in \mathbb{R}^{n^{(1)} \times n^{(2)} \ldots \times n^{(d)}}$$

when we assume that the indices are fixed (not under permutations).

Let us consider that for each $k \in D$ we have a vector space $V^{(k)}$ with a basis

$$B^{(k)} = \left(\mathbf{e}_1^{(k)}, \mathbf{e}_2^{(k)}, \ldots, \mathbf{e}_{n^{(k)}}^{(k)} \right).$$

A vector of $V^{(k)}$ can be written

$$\mathbf{v}^{(k)} = v_1^{(k)} \mathbf{e}_1^{(k)} + v_2^{(k)} \mathbf{e}_2^{(k)} + \ldots + v_{n^{(k)}}^{(k)} \mathbf{e}_{n^{(k)}}^{(k)}.$$

Before we introduce other notations, let us define the *tensor product*, known as the *Kronecker product* that has been recalled in (1.21) for general matrices. We have to emphasize that this operation is among the most useful in numerical tensor calculation. For two vectors $\mathbf{a} \in \mathbb{R}^{n^{(1)}}$ and $\mathbf{b} \in \mathbb{R}^{n^{(2)}}$, we have to distinguish two different but equivalent uses of Kronecker product. The first results in a long vector whose length is the product of lengths of inputs.

$$\text{For} \quad \mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n^{(1)}} \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n^{(2)}} \end{pmatrix}, \quad \mathbf{a} \otimes \mathbf{b} = \begin{pmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ \vdots \\ a_{n^{(1)}} \mathbf{b} \end{pmatrix} \in \mathbb{R}^{n^{(1)} . n^{(2)}}. \tag{5.2}$$

The second use of the tensor product is some times called *outer product*. It results in a rank-one matrix of size $n^{(1)} \times n^{(2)}$:

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^T = \begin{pmatrix} a_1 \mathbf{b}^T \\ a_2 \mathbf{b}^T \\ \vdots \\ a_{n^{(1)}} \mathbf{b}^T \end{pmatrix} \in \mathbb{R}^{n^{(1)} \times n^{(2)}}. \tag{5.3}$$

Whenever there is no other specific indications, the notation $\otimes$ will refer to this last tensor product. Observe that the product of two tensors of order one (vectors) results in a tensor of order two (matrix). Orders have been added up.

The notation

$$\otimes_{k=1}^{d} \mathbf{v}^{(k)} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \otimes \ldots \otimes \mathbf{v}^{(d)}$$

indicates the Kronecker product of d vectors

$$\mathbf{v}^{(1)} \in V^{(1)}, \mathbf{v}^{(2)} \in V^{(2)}, \ldots, \mathbf{v}^{(d)} \in V^{(d)}.$$

Finally, we consider a numerical tensor $\mathbf{V} \in \mathcal{V}$ as a multidimensional array whose numerical values are

$$v_{j_1 j_2 \cdots j_d} = \mathbf{V}(j_1, j_2, ..., j_d),$$

where each index j_k varies up to the number $n^{(k)}$ that is the dimension of its associated vector space $V^{(k)}$.

5.1.2 Visual representation

A numerical tensor can also be seen as an array that is a high-order generalization of scalars (tensor of order $d = 0$), vectors (tensor of order $d = 1$), matrices (tensor of order $d = 2$), while a tensor of order $d > 2$ is called simply a tensor (see [82]).

Figure 5.1 illustrates visual representation of numerical tensors of order $d = 0, 1, 2, 3, 4, 5$ that are respectively scalar, vector, matrix, 3^{rd}-order tensor, 4^{th}-order tensor and 5^{th}-order tensor. Two ways of representing tensors are presented. Either by a dot with lines that represent the directions or by boxes and slices. Tensors of order $d > 3$ can also be represented as a stack of volumes. A 0-order tensor is a scalar (5.1a), a 1-order tensor is a vector (5.1b), matrices are 2-order tensors (5.1c), a 3-order tensor is visualized in (5.8b) as a stack of slices (matrices), 5-order tensor is visualized in (5.1e) as a stack of 3-order tensors and in (5.1g) as a stack of stacks of slices. A more general representation of a 5-order tensor is shown in (5.1f) while a 4-order tensor is highlighted as a part of the 5-order tensor in (5.1e).

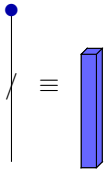
A vector can be expressed in *row mode* or in *column mode*. For matrices, columns may refer to the *first mode* while rows refer to the *second mode* (the inverse can be also assumed). The concept of *mode* allows to generalize these concepts of rows, columns, axis or directions for high-dimensional tensors. In 3D, we consider the lexicographic ordering shown in Figure 5.2.

Comparatively to matrices, the *mode-n* vectors of a tensor are called *fibers*. They are obtained by fixing all but one index. For example, a fiber of a 4-order tensor $\mathbf{V}$ can be written

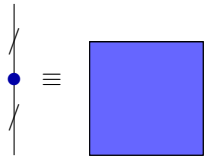
$$\mathbf{a} = \mathbf{V}(:, :, j_3, :),\ j_3 = 1, 2, \ldots, n^{(3)}.$$

Note that, in a tensor of order d, one can define a *k-th order slice* ($1 \leq k \leq d$) by fixing all but k indices (see [8], for examples).

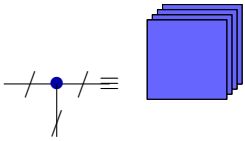
(a) Scalar $a(d=0)$



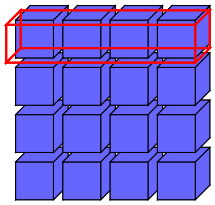
(b) Vector $\mathbf{a}(d=1)$



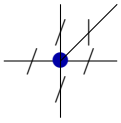
(c) Matrix A $(d=2)$



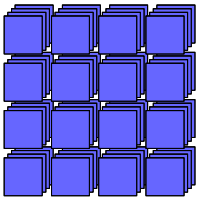
(d) Tensor $\mathbf{A}$ $(d=3)$



(e) Tensor $\mathbf{A}$ $(d=5)$. The red contour highlights a 4-order tensor.



(f) Tensor $\mathbf{A}$ $(d=5)$



(g) Tensor $\mathbf{A}$ $(d=5)$ in terms of block matrices

Figure 5.1: Visual representation of numerical tensors. tensors of order $d = 0, 1, 2, 3, 4, 5$ that are respectively scalar, vector, matrix, 3^{rd}-order tensor, 4^{th}-order tensor and 5^{th}-order tensor.

Let us now define a tensor space. Following Hackbush [6, p.57], one way of defining a d-dimensional tensor space is as follows.

Definition 5.1 (*Tensor space*)

Let be given $d+1$ vector spaces $V^{(1)}, V^{(2)}, \ldots, V^{(d)}$, of respective dimensions $n^{(1)}, n^{(2)}, \ldots, n^{(d)}$ and $\mathcal{V}$ on a common field $\mathbb{K}$. Consider

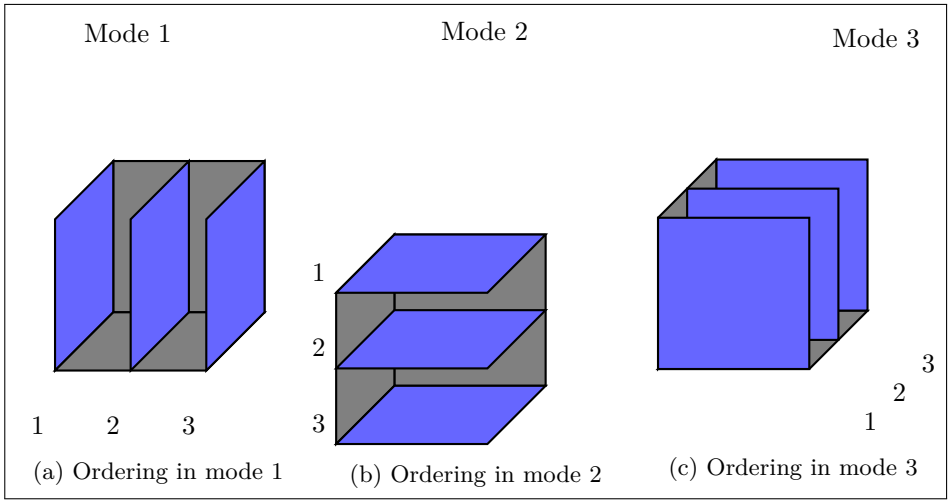
$$B^{(k)} = \left(\mathbf{e}_{j_k}^{(k)}\right), \ 1 \leq j_k \leq n^{(k)},$$

the respective bases of the d vector spaces $\left(V^{(k)}\right)$, $k = 1 : d$. The mapping

$$\otimes : V^{(1)} \times V^{(2)} \times \ldots \times V^{(d)} \longrightarrow \mathcal{V}, \tag{5.4}$$

is a tensor product and $\mathcal{V}$ *is an algebraic tensor space if these properties hold:*

1. *span property:* $\mathcal{V} = span\left\{\otimes_{k=1}^{d} \mathbf{v}^{(k)}, \quad \mathbf{v}^{(k)} \in V^{(k)}\right\}$;



(a) Ordering in mode 1 (b) Ordering in mode 2 (c) Ordering in mode 3

Figure 5.2: Consideration of modes and ordering in $3D$

2. $\otimes$ *is multilinear: for all* $\lambda \in \mathbb{K}$, $\mathbf{v}^{(k)}, \mathbf{w}^{(k)} \in V^{(k)}$, *and* $k \in \{1, 2, \ldots, d\}$

$$\mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \otimes \ldots \otimes \left(\lambda \mathbf{v}^{(k)} + \mathbf{w}^{(k)}\right) \otimes \ldots \otimes \mathbf{v}^{(d)} =$$

$$\lambda \left(\mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \otimes \ldots \otimes \mathbf{v}^{(k)} \otimes \ldots \otimes \mathbf{v}^{(d)}\right) + \left(\mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \otimes \ldots \otimes \mathbf{w}^{(k)} \otimes \ldots \otimes \mathbf{v}^{(d)}\right);$$

3. *the linearly independent vectors* $\left\{\mathbf{v}_{j_k}^{(k)}\right\} \subset V^{(k)}$ *imply linearly independent vectors* $\left\{\otimes_{k=1}^{d} \mathbf{v}_{j_k}^{(k)}\right\} \subset \mathcal{V}$, *where* $j_k \in I_k = \{1, ..., n^{(k)}\}$ *and* $n^{(k)}$ *is the dimension of the vector space* $V^{(k)}$.

Then, the space $\mathcal{V}$ *equipped with this mapping (this tensor product) is called algebraic tensor space or simply tensor space and is sometimes denoted* $\mathcal{V} = a\bigotimes_{k=1}^{d} V^{(k)}$ *(a, for algebraic). Finally, the dimension of the tensor space is*

$$dim(\mathcal{V}) = \prod_{k=1}^{d} dim(V^{(k)}).$$

If, in addition, each of the d vector spaces is a Banach space, with a norm $\|.\|_{V^{(k)}}$, the space $\mathcal{V}$ is a *topological tensor space* sometimes denoted $\mathcal{V} = \|.\| \bigotimes V^{(k)}$. In finite dimension, the normed spaces equipped with the tensor product are Banach tensor spaces and the distinction between the topological and algebraic tensor spaces is omitted. In the following, unless otherwise stated, $\mathcal{V} = \bigotimes_{k=1}^{d} V^{(k)}$ is a tensor space of finite dimension. It is generated by vector spaces of general form $V^{(k)} = \mathbb{R}^{I_k}$, $k = 1 : d$.

To make a link with matrices, tensors are considered as mappings from linear spaces to other linear spaces whose coordinates transform multilinearly under

a change of bases [85].

5.1.3 Tensor in full format

Let $\left(V^{(k)}\right)$, $k = 1 : d$ be vector spaces and $B^{(k)} = \left(\mathbf{e}_{j_k}^{(k)}\right)$, $1 \leq j_k \leq n^{(k)}$ be their respective bases. A tensor $\mathbf{V}$, element of a tensor space $\mathcal{V}$ generated by vector spaces $V^{(k)}$, can be written in the form:

$$\mathbf{V} = \sum_{j_1=1}^{n^{(1)}} \dots \sum_{j_d=1}^{n^{(d)}} v_{j_1 j_2 \dots j_d} \otimes_{k=1}^{d} \mathbf{e}_{j_k}^{(k)}, \quad v_{j_1 j_2 \dots j_d} \in \mathbb{R}. \tag{5.5}$$

The tensor is said to be in full format when it is defined by all the coefficients $\left(v_{j_1 j_2 \dots j_d}\right)_{1 \leq j_k \leq n^{(k)},\ 1 \leq k \leq d}$. In other words, it is represented as a d-dimensional array with all its entries. From the relation (5.5), a basis $\mathbf{B}$ of $\mathcal{V}$ is defined by the tensor product of vector space bases as $\mathbf{B} = B^{(1)} \times \dots \times B^{(d)}$ and $dim(\mathcal{V}) = \prod_{k=1}^{d} n^{(k)}$.

Figure 5.3 illustrates a three dimensional tensor as a 3-dimensional array with its scalar entries (left), the same tensor is presented as a stack of fibers (middle) and as a stack of matrices or $2D$-slices (right).

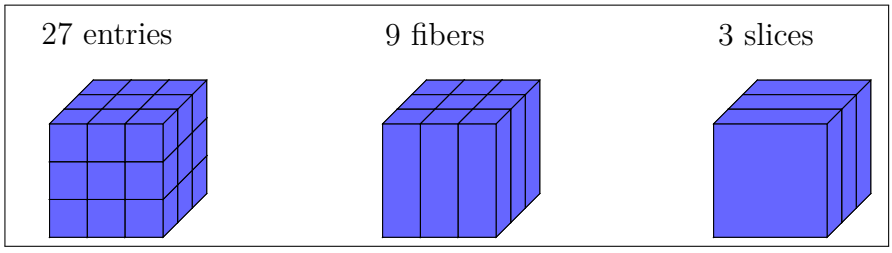


Figure 5.3: A $3D$ tensor in *full format* can be seen as $3 \times 3 \times 3$ array (left) or as 3×3 fibers (middle), where each fiber is a vector of 3 entries, or as a stack of 3 slices (right) where each slice is a matrix of order 3×3.

5.1.4 Some numerical tensor treatments

Vectorization of tensors

Matrices and multidimensional arrays can be easily reshaped in a vector by ordering their entries in long columns following the so-called *lexicographical ordering*[1]. Given a tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_d}$, where each entry is determined by a d-tuple $(j_1, j_2, \dots, j_d)$, a natural number $N = \prod_{k=1}^{d} \#I_k$ and an index set $J = \{1, 2, \dots, N\}$, one can obtain a long vector $\mathbf{a}$ from this tensor, where each entry is determined by one index $j \in J$, by defining an isomorphism (see [86])

$$\phi : (j_1, j_2, \dots, j_d) \in I_1 \times I_2 \times \dots \times I_d \longmapsto J$$

1. Different program languages may use different lexicographical orderings [6, 157].

such that

$$j = \phi(j_1, j_2, \ldots, j_d) = j_1 + \sum_{k=2}^{d} (j_k - 1) \prod_{l=1}^{k-1} n^{(l)}. \tag{5.6}$$

In practice, the indices in J can be obtained as

$$J = \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_d \end{bmatrix} \in \mathbb{R}^{n^{(1)}.n^{(2)}.\ldots.n^{(d)}}.$$

Matricization of tensors

The *matricization* known also as *unfolding* or *flattening* is an operation of reshaping a tensor into a matrix. A tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ can be transformed into a matrix in different ways. The idea is to shrink the dimension-set $D = \{1, 2, \ldots, d\}$ into 2 disjoint and complementary subsets t and t^c such that $\emptyset \subsetneq t \subsetneq D$ and $t^c = D \backslash t$. Then, one gets index sets $\mathbf{I}_t = \times_{k \in t} I_k$ (that will constitute row indices) and $\mathbf{I}_{t^c} = \times_{k \in t^c} I_k$ (that will constitute column indices). Then the t- *matricization* of $\mathbf{A}$ is given by

$$M_t(\mathbf{A}) = A^{(t)} \in \mathbb{R}^{\mathbf{I}_t \times \mathbf{I}_{t^c}}, \tag{5.7}$$

where M_t is considered as a t-matricization mapping. The entries of the matrix $A^{(t)}$ are $A^{(t)}(p, q)$ where the indices p and q are given by

$$\begin{aligned} p &= j_1 + \textstyle\sum_{r=2}^{k} (j_r - 1) \prod_{l=1}^{r-1} n^{(l)} \\ q &= j_{k+1} + \textstyle\sum_{r=k+2}^{d} (j_r - 1) \prod_{l=1}^{r-1} n^{(l)}. \end{aligned} \tag{5.8}$$

The t-rank is the rank of the matrix $A^{(t)}$.

For t that is a singleton, one forms the so-called *mode-k matrices* or *mode k unfolding matrices* given by

$$A^{(k)} \in \mathbb{R}^{I_k \times \left(I_1 \times I_2 \times \ldots \times I_{k-1} \times I_{k+1} \times \ldots \times I_d\right)}, \quad k = 1 : d. \tag{5.9}$$

Observe that, it suffices to reshape a matrix by fixing the index corresponding to the index-set I_k in row-mode while varying all the other indices in the column-mode. The matrix $A^{(k)}$ is equivalent to the whole tensor but it is a matrix in mode k. The entry $a_{j_1 j_2 \ldots j_d}$ becomes a_{il} where $i = j_k$ and

$$l = j_1 + \textstyle\sum_{r=2}^{k-1} (j_r - 1) \prod_{l=1}^{r-1} n^{(l)} + \sum_{r=k+1}^{d} (j_r - 1) \prod_{l=1}^{r-1} n^{(l)}. \tag{5.10}$$

Example

Given a 3-order tensor $\mathbf{A} \in \mathbb{R}^{3 \times 3 \times 3}$ with $n^{(1)} = n^{(2)} = n^{(3)} = 3$, its size is $n = [3, 3, 3]$ and its number of entries is $N = 27$. Figure 5.4 illustrates the indices ordering considered here for a 3-dimensional tensor. The figure shows that for different modes, indices designates different thinks but the whole tensors is described by any mode.

When vectorized, its entries will be positioned from 1 to 27 (one may use the *vec* command of `Matlab`). The matricization of $\mathbf{A}$ in the respective modes $1, 2, 3$ (see [87, p.220]), gives the following 3 index matrices.

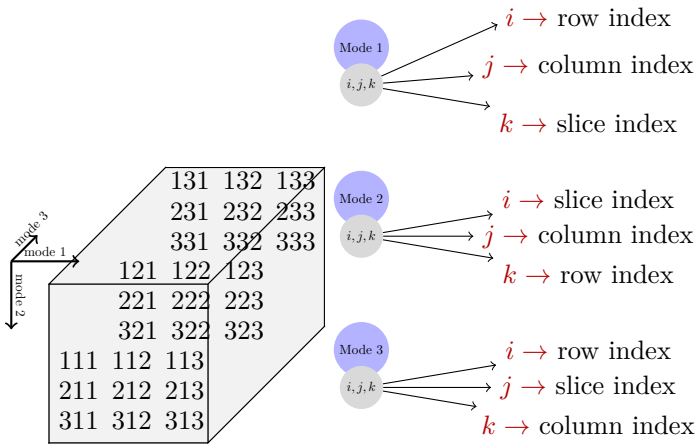


Figure 5.4: Indices in a 3-order tensor.

$$
A^{(1)} = \begin{array}{cccccccccc}
\mathbf{111} & \mathbf{112} & \mathbf{113} & 121 & 122 & 123 & 131 & 132 & 133 \\
\mathbf{211} & \mathbf{212} & \mathbf{213} & 221 & 222 & 223 & 231 & 232 & 233 \\
\mathbf{311} & \mathbf{312} & \mathbf{313} & 321 & 322 & 323 & 331 & 332 & 333
\end{array}
$$

$$
A^{(2)} = \begin{array}{cccccccccc}
\mathbf{111} & \mathbf{112} & \mathbf{113} & 211 & 212 & 213 & 311 & 312 & 313 \\
\mathbf{121} & \mathbf{122} & \mathbf{123} & 221 & 222 & 223 & 321 & 322 & 323 \\
\mathbf{131} & \mathbf{132} & \mathbf{133} & 231 & 232 & 233 & 331 & 332 & 333
\end{array}
$$

$$
A^{(3)} = \begin{array}{cccccccccc}
\mathbf{111} & \mathbf{121} & \mathbf{131} & 211 & 221 & 231 & 311 & 321 & 331 \\
\mathbf{112} & \mathbf{122} & \mathbf{132} & 212 & 222 & 232 & 312 & 322 & 332 \\
\mathbf{113} & \mathbf{123} & \mathbf{133} & 213 & 223 & 233 & 313 & 323 & 333
\end{array}
$$

Observe that, for each $k = 1, 2, 3$, the *row number* of matrix $A^{(k)}$ corresponds to the $\mathbf{k^{th}}$ index of the tensor $\mathbf{A}$.

Figure 5.5 visualizes a 3-dimensional tensor of size $(3 \times 3 \times 3)$ and its unfoldings in mode 1, mode 2 and mode 3 respectively. The number $r_k, k = 1, 2, 3$ denotes the rank the the unfolding matrix in mode k.

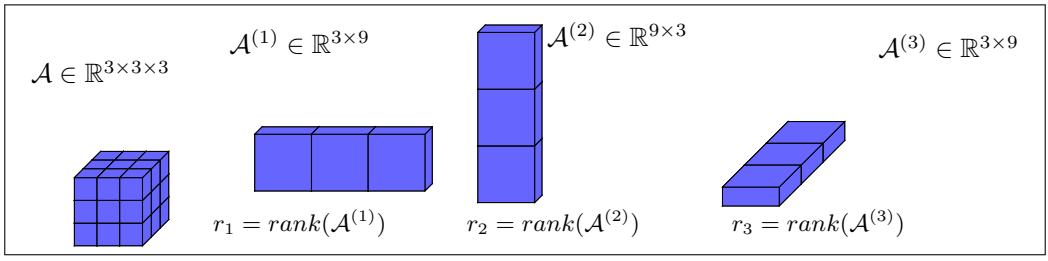


Figure 5.5: From left to right: tensor of order $d = 3$, size $(3 \times 3 \times 3)$ and it's matricization in mode 1, mode 2, and mode 3, respectively.

Tensorization of vectors

The tensorization of a vector is the opposite operation to the vectorization of a tensor. Since the relation (5.6) is a bijection, its reciprocal can be used to tensorize vectors. Consider a vector $\mathbf{x} \in \mathbb{R}^N$ where $N = \prod_{k=1}^{d} n^{(k)}$ for some $n^{(k)} \in \mathbb{N}$, the tensorization of $\mathbf{x}$ leads to a d-dimensional tensor $\mathbf{X}$ of size $[n^{(1)}, \ldots, n^{(d)}]$ such that

$$\mathbf{X}(j_1, j_2, \ldots, j_d) = \mathbf{x}_j$$

where j is given by the relation (5.6).

Tensor-matrix product

We follow [88] for the definitions below. To introduce the general notation of tensor-matrix product, we need to recall the principles of matrix-matrix product, denoted here by the dot. Let be given matrices $F \in \mathbb{R}^{I_1 \times I_2}, U \in \mathbb{R}^{I_1 \times J_1}$ and $V \in \mathbb{R}^{I_2 \times J_2}$. Then we can compute the product

$$G = U^T.F.V \in \mathbb{R}^{J_1 \times J_2} \tag{5.11}$$

obeing the compatibility of the mode sizes. Analizing the product (5.11) [88] one can note that the rows of F are multiplied by U (not U^T) while the columns of F are multiplied by V. In the same way, it can be noticed [2] that, in the matrix result G, the rows of U are associated to the row space of G while the rows of V are associated to the column space of G. Thus, one may avoid to transpose U but work with the so-called mode-k multiplication $\times_k$ such that

$$G = F \times_1 U \times_2 V. \tag{5.12}$$

The notation in (5.12) means that the matrix F is mulplied by U in mode 1 (row mode) and by V in mode-2 (columns mode).

In general, the k-mode multiplication is given in the definition below (see [88])

Definition 5.2

Let be given a tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ and a matrix $M \in \mathbb{R}^{I_k \times J_k}$, $1 \leq k \leq d$. The k-mode product of $\mathbf{A}$ and M denoted $\mathbf{A} \times_k M$ is a $(I_1 \times \ldots \times I_{k-1} \times J_k \times I_{k+1} \ldots I_d)$ tensor whose entries are given by

$$(\mathbf{A} \times_k M)(i_1, i_2, \ldots, i_{k-1}, j_k, i_{k+1}, \ldots, i_d) \stackrel{\text{def}}{=} \sum_{i_k} a_{i_1 i_2 \ldots i_{k-1} i_k i_{k+1} \ldots i_d} m_{i_k j_k}. \tag{5.13}$$

Given then tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ and matrices $F \in \mathbb{R}^{I_k \times J_k}, G \in \mathbb{R}^{I_l \times J_l}$. The k-mode product verifies the properties:

$$(\mathbf{A} \times_k F) \times_l G = (\mathbf{A} \times_l G) \times_k F = \mathbf{A} \times_k F \times_l G. \tag{5.14}$$

Given the tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ and matrices $F \in \mathbb{R}^{I_k \times J_k}$ and $G \in \mathbb{R}^{J_k \times K_k}$.

$$(\mathbf{A} \times_k F) \times_k G = \mathbf{A} \times_k (G.F). \tag{5.15}$$

2. In [88] the mode-1 represents the columns mode while here this is represented by rows mode.

Tensor-tensor product

In general, given tensors $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_{d_1}}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_{d_2}}$, their product gives a tensor $\mathbf{C}$ of order $d_1 + d_2$ such that

$$\mathbf{C}\left(i_1, i_2, \ldots i_{d_1}, j_1, j_2, \ldots, j_{d_2}\right) = \mathbf{A}\left(i_1, i_2, \ldots i_{d_1}\right)\mathbf{B}\left(j_1, j_2, \ldots, j_{d_2}\right) = \mathbf{A} \otimes \mathbf{B} \quad (5.16)$$

For example, from the product $\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} \otimes \begin{pmatrix}1\\1\end{pmatrix}$ we get a 3-dimensional tensor $\mathbf{A} \in \mathbb{R}^{2\times 2\times 2}$ that has two identical slices $\begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}$ and may be given by its mode-1 unfolding

$$A^{(1)} = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 \end{array}\right]$$

Khatri-Rao product

Another product from the Kronecker product is the *Khatri-Rao product*. Consider the matrices $A = [\mathbf{a}_1, ..., \mathbf{a}_r] \in \mathbb{R}^{n\times r}$ and $B = [\mathbf{b}_1, ..., \mathbf{b}_r] \in \mathbb{R}^{m\times r}$ where $\mathbf{a}_k$ and $\mathbf{b}_k$ denote the k^{th} column of A and B respectively. The Khatri-Rao product of A and B is the matrix

$$C = A \odot B \in \mathbb{R}^{nm\times r} \quad (5.17)$$

whose columns are the Kronecker product (5.2) of respective columns of A and B. That is

$$\mathbf{c}_i = \mathbf{a}_i \otimes \mathbf{b}_i,\ i = 1 : r.$$

Contracted product

The multiplication of tensors can be generalized simply by the so- called *contracted product*. The contraction can be done on a specific mode (*mode-k product*) or on multiple modes: for two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{n^{(1)}}$ the mode-1 product is simply the classical scalar product

$$\mathbf{u}.\mathbf{v} = \sum_{i=1}^{n^{(1)}} \mathbf{u}_i \mathbf{v}_i.$$

For a matrix $A \in \mathbb{R}^{n^{(1)}\times n^{(2)}}$ and vectors $\mathbf{u} \in \mathbb{R}^{n^{(1)}}, \mathbf{v} \in \mathbb{R}^{n^{(2)}}$ we can use the following products to express matrix-vector products:

$$\begin{array}{lcl} A\mathbf{v} & = & A \times_1 \mathbf{v} \in \mathbb{R}^{n^{(1)}} \quad \text{mode-1 product,}\\ \mathbf{u}^T A & = & A \times_2 \mathbf{u} \in \mathbb{R}^{n^{(2)}} \quad \text{mode-2 product,}\\ \mathbf{u}^T A\mathbf{v} & = & A \times_1 \mathbf{v} \times_2 \mathbf{u} \in \mathbb{R} \quad \text{mode-1-mode-2 product.} \end{array}$$

Observe that on the last multiplication the contraction is made on the two modes and the result is a scalar. Thus, given two tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_{d_1}}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_{d_2}}$, if there is $I_k = J_k,\ 1 \le k \le d_1, d_2$ then the *mode-k product* is given by

$$\mathbf{C} = (\mathbf{A} \times_k \mathbf{B}) = \sum_{j_k=1}^{n^{(k)}} \mathbf{A}_{i_1,\ldots,i_k,\ldots i_{d_1}} \mathbf{B}_{j_1,\ldots,j_{k-1},j_k,j_{k+1}\ldots,j_{d_2}} \equiv \mathbf{A} \times_k B^{(k)}. \quad (5.18)$$

The product (5.18) is the general *contracted product* where the corresponding directions are contracted and the resulting tensor $\mathbf{C}$ is of order $d = d_1 + d_2 - 2$. The full multilinear product contracts all the directions and writes

$$\mathbf{C} = \langle \mathbf{A}, \mathbf{B} \rangle = \mathbf{A} \times_1 B^{(1)} \times_2 B^{(2)} \dots \times_k B^{(k)}$$

that results in a scalar. This is the tensor inner product. Of course, the number of indices in corresponding directions have to be equal.

The Kronecker product can be generalized to operators, especially to separable differential operators [89]. The interesting tasks now include efficient storage of tensors and relevant mathematical operations for efficient computation of these stored tensors. Answers to these tasks have lead to certain tensor formats. For PDEs, one way is to specify some basis functions and then store the coefficients in these bases representations [89]. Before we address the most known representation formats, note that reliable representation of data may lead to significant saving of storage memory and operations. This is illustrated by the example below from [6, p. 171].

Example 5.1

Consider the index set $I_0 = \{0, 1, 2, \dots, N-1\}$ *and a vector* $(\mathbf{x})_i = v^i, \quad i \in I_0$, *and* $v \in \mathbb{R}$, *with* $N = \prod_{k=1}^{d} n^{(k)}$, *for some* $n^{(k)} \in \mathbb{N}, n^{(k)} \geq 2$. *For* $d = 2$, *the matricization of the vector* $\mathbf{x}$ *gives the matrix*

$$M = \begin{pmatrix} v^0 & v^{n^{(1)}} & \dots & v^{(n^{(2)}-1)n^{(1)}} \\ v^1 & v^{n^{(1)}+1} & \dots & v^{n^{(2)}-1n^{(1)}+1} \\ \vdots & \vdots & \ddots & \vdots \\ v^{n^{(1)}-1} & v^{2n^{(1)}-1} & \dots & v^{n^{(2)}n^{(1)}-1} \end{pmatrix} = \begin{pmatrix} v^0 \\ v^1 \\ v^2 \\ \vdots \\ v^{n^{(1)}-1} \end{pmatrix} \begin{pmatrix} v^0 \\ v^{n^{(1)}} \\ v^{2n^{(1)}} \\ \dots \\ v^{(n^{(2)}-1)n^{(1)}} \end{pmatrix}^T.$$

The last equality shows that M *is a rank-one matrix that can be written using the tensor product and saving significant memory:*

$$M = \begin{pmatrix} v^0 \\ v^1 \\ v^2 \\ \vdots \\ v^{n^{(1)}-1} \end{pmatrix} \otimes \begin{pmatrix} v^0 \\ v^{n^{(1)}} \\ v^{2n^{(1)}} \\ \dots \\ v^{(n^{(2)}-1)n^{(1)}} \end{pmatrix}.$$

The same calculation can be made for $d = 3$ *getting*

$$T = \begin{pmatrix} v^0 \\ v^1 \\ v^2 \\ \vdots \\ v^{n^{(1)}-1} \end{pmatrix} \otimes \begin{pmatrix} v^0 \\ v^{n^{(1)}} \\ v^{2n^{(1)}} \\ \vdots \\ v^{(n^{(2)}-1)n^{(1)}} \end{pmatrix} \otimes \begin{pmatrix} v^0 \\ v^{n^{(1)}n^{(2)}} \\ v^{2n^{(1)}n^{(2)}} \\ \dots \\ v^{(n^{(3)}-1)n^{(2)}n^{(1)}} \end{pmatrix}.$$

The next proposition can be proved by induction following the example above.

Proposition 5.3

Consider $N = n^{(1)}n^{(2)}\dots n^{(d)}$ *with* $n^{(k)} \geq 2$, *for all* $1 \leq k \leq d$, *and set* $I_0 = \{0, 1, 2, \dots, N-1\}$. *The vector* $(\mathbf{x})_i = \left(v^i\right)$, $i \in I_0$ *and* $v \in \mathbb{R}$ *corresponds to the elementary tensor* $\mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \otimes \dots \otimes \mathbf{v}^{(d)}$ *where* $\mathbf{v}^{(k)} \in \mathbb{R}^{I_k}$ *and is defined by*

$$\mathbf{v}^{(k)} = \begin{pmatrix} v^0 \\ v^{p^{(k)}} \\ v^{2p^{(k)}} \\ \vdots \\ v^{(n^{(k)}-1)p^{(k)}} \end{pmatrix} \quad \text{with} \quad p^{(k)} = \prod_{l=1}^{k-1} n^{(l)}.$$

In the case where,$n^{(k)} = 2$ *for all* $k \in \{1, 2, \dots, d\}$, *that is,* $N = 2^d$ *and* $p^{(k)} = 2^{k-1}$, *then* $\mathbf{v}^{(k)} = \begin{pmatrix} 1 \\ v^{2^{k-1}} \end{pmatrix}$ *and the data size is reduced from* N *to* $2d = 2logN$.

In addition to the saving of memory storage enabled by appropriate representation of tensors, such representations may also lead to more stable algorithms [6, p.172]. Thus, it is important to investigate different representations of tensors. This requires the definition of elementary tensors.

Definition 5.4 (*Elementary tensor*)

An elementary tensor is any tensor of $\mathcal{V} = \bigotimes_{k=1}^{d} V^{(k)}$ *that can be written as:* $\mathbf{V} = \otimes_{k=1}^{d} \mathbf{v}^{(k)}$, $\mathbf{v}^{(k)} \in V^{(k)}$. *In other words, any tensor that can be written as a single tensor product of vectors is qualified elementary tensor. The set of elementary tensors in* $\mathcal{V}$ *is denoted here by*

$$\mathcal{C}_1(\mathcal{V}) = \left\{ \otimes_{k=1}^{d} \mathbf{v}^{(k)},\ \mathbf{v}^{(k)} \in V^{(k)} \right\}. \tag{5.19}$$

Any tensor can be approximated using elementary tensors. Let $\mathbf{V}$ be a a d-dimensional tensor from $\mathcal{V}$. Then, it can be written

$$\mathbf{V} = \sum_{j=1}^{r} \alpha_j \otimes_{k=1}^{d} \mathbf{v}_j^{(k)}, \quad \alpha_j \in \mathbb{R}, \quad \mathbf{v}_j^{(k)} \in V^{(k)}. \tag{5.20}$$

Here, the tensor is considered as a linear combination of elementary tensors. This allows us to introduce the concept of *tensor rank* and other tensor representations or formats. However, notice that for tensors, the concept of rank is not uniquely defined. Below we present certain concepts denoting tensor rank.

Definition 5.5 (*Canonical tensor rank*)

Let $\mathbf{V}$ *be a a d-dimensional tensor. The smallest integer such that the tensor* $\mathbf{V}$ *can be expressed by* (5.20) *is called the* canonical tensor rank *of* $\mathbf{V}$. *In other words, it is the smallest number of elementary tensors used to represent the tensor.*

Note that, unlike matrix rank, tensor rank may exceed all the dimension sizes. In addition, for tensors of order $d > 2$, the tensor rank may depend on the field (weither

real or complex) [90]. For example, consider tensors $\mathbf{A}, \mathbf{G} \in \mathbb{R}^{2\times 2\times 2}$ defined by their mode-1 unfoldings (see 5.9):

$$A^{(1)} = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

and

$$G^{(1)} = \left[\begin{array}{cc|cc} 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{array}\right].$$

In the real field, the tensor $\mathbf{A}$ has rank 1 because it can be written

$$\mathbf{A} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Conversely, in the real field the tensor $\mathbf{G}$ needs three elementary tensors:

$$\mathbf{G} = 4\begin{pmatrix} 1 \\ 0 \end{pmatrix}^{\otimes 3} + \begin{pmatrix} -1 \\ -1 \end{pmatrix}^{\otimes 3} + \begin{pmatrix} -1 \\ 1 \end{pmatrix}^{\otimes 3}.$$

where the notation $x^{\otimes k} = \overbrace{x \otimes x \ldots \otimes x}^{k\, factors}$ is the tensor product of identical k factors. However, if the decomposition in the complex field is allowed, by considering $\sqrt{-1} = i$, we may write

$$\mathbf{G} = \begin{pmatrix} 1 \\ i \end{pmatrix}^{\otimes 3} + \begin{pmatrix} 1 \\ -i \end{pmatrix}^{\otimes 3}.$$

We conclude that the tensor $\mathbf{G}$ has canonical tensor rank 3 in $\mathbb{R}$ and 2 in $\mathbb{C}$. From this example, one can observe that imposing the decomposition to be in $\mathbb{R}$, may increase the rank. This situation is general and due to this, many ranks have been defined, such as *nonnegative rank*, *symmetric rank*, *border rank*, *structured rank*,*Tucker rank*, *hierarchical Tucker rank*, *Tensor Train rank* or *core tensor rank*. We refer to [85] for these concepts while some of them are subsequently defined.

In addition to the canonical tensor rank, we can define a multilinear rank as follows.

Definition 5.6 (*Multilinear rank*)

Given a tensor $\mathbf{A}$ *of order* d*, its multilinear rank denoted* ml-rank($\mathbf{A}$) *is the* d*-tuple* $(r_1, r_2, \ldots, r_d) = \left(rank(A^{(1)}), rank(A^{(2)}), \ldots, rank(A^{(d)})\right)$*, where each* r_k *is the number of linearly independent columns (or rows) vectors of the matrix obtained after matricization of the tensor in mode* k*. That is*

$$r_k, = rank(A^{(k)}), \quad k = 1, 2, \ldots, d. \tag{5.21}$$

5.2 Tensor formats and tensor decompositions

Currently, in addition to the *full format* introduced in the previous section, four tensor formats are the most commonly used in *numerical tensor analysis* (see [6]). These formats are representations of the tensor in a way that allows an efficient processing of its elements.

We give a brief description of three of them in this section and a fourth is further developed in the next section, and will be used more specifically. We refer to [6] and [8] for more details.

5.2.1 Canonical Polyadic (CP) format

A tensor is given under *canonical format*, or *polyadic canonical format*, when it is written as a linear combination of R elementary tensors. That is, a tensor $\mathbf{F} \in \mathcal{V} = \otimes_{k=1}^{d} V^{(k)}$ where $V^{(k)} = \mathbb{R}^{I_k}$ is written as

$$\mathbf{F} = \sum_{i=1}^{R} g_i \mathbf{u}_i^{(1)} \otimes \mathbf{u}_i^{(2)} \ldots \otimes \mathbf{u}_i^{(d)}, \tag{5.22}$$

where R is the canonical rank of $\mathbf{F}$ and $\mathbf{u}_i^{(k)}$, $i = 1, \ldots, n^{(k)}$ are vector from vector space $V^{(k)}$ of dimension $n^{(k)}$. Since a scalar product is assumed to be defined on the considered tensor space, one may impose the vectors $\mathbf{u}_i^{(k)}, k = 1 : d, \quad i = 1 : n^{(k)}$ to be of unite norm. In this context, scalars $g_i, \quad i = 1 : R$ form a d-dimensional array $\mathbf{G}$ whose matricization in any mode is diagonal. Thus, the decomposition can be equivalently written in matrix format

$$\mathbf{F} = \mathbf{G} \times_1 U^{(1)} \times_2 U^{(2)} \ldots \times_d U^{(d)}, \tag{5.23}$$

where $\mathbf{G} = diag_d(g_1, g_2, ..., g_R)$ is the *core tensor* that is d-diagonal. This means that it is diagonal in a d-dimensional space (its matricization in any mode is diagonal). The factors $(U^{(k)})_{k=1:d}$ are factor matrices built with the vectors $\mathbf{u}_i^{(k)}$ and the notation $\times_k$ represents the mode-k product (5.12). Using the *Khatri-Rao product* $\odot$ and matricization of the tensor $\mathbf{F}$, the relation (5.23) writes :

$$F^{(k)} = U^{(k)} G^{(k)} (U^{(d)} \odot ... \odot U^{(k+1)} \odot U^{(k-1)} ... \odot U^{(1)})^T, \; k = 1 : d, \tag{5.24}$$

where $F^{(k)}$ is the matrix representation of the tensor in mode k.

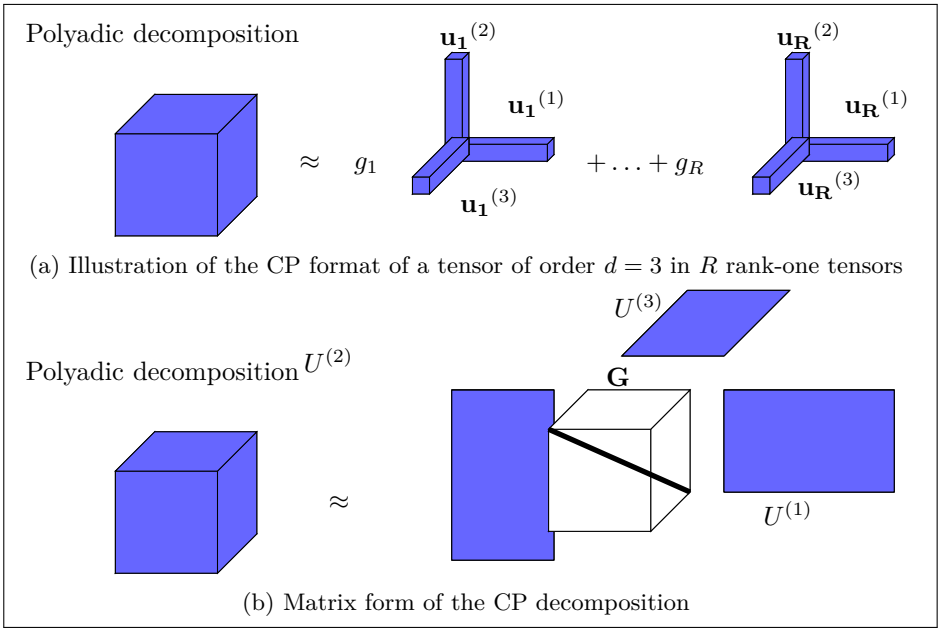
Figure 5.6 illustrates the canonical polyadic decomposition of a 3-order tensor, where the decomposition is presented as a linear combination of rank-one tensors. The core tensor $\mathbf{G}$ is a d-diagonal matrix.

This format has the advantage that the core tensor is diagonal but a significant disadvantage is that the matrix factors $(U^{(k)})_{k=1:d}$ are not necessarily orthogonal, and the rank R may be too large. One may impose an orthogonality constraint on the matrix factors while the the core tensor might not still diagonal. This is the idea of Tucker format.

5.2.2 Tucker format

A tensor is said to be in a *Tucker format* when it is written as a product of a *core tensor*, smaller in size than the initial tensor, with orthogonal matrix factors. Hence, given a tensor $\mathbf{F} \in \mathcal{V} = \otimes_{k=1}^{d} V^{(k)}$ where $V^{(k)} = \mathbb{R}^{I_k}$ with $n^{(k)} = \#I_k$ large and $r_k < n^{(k)}, k = 1 : d$, one writes:

$$\mathbf{F} = \sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \ldots \ldots \sum_{i_d=1}^{r_d} g_{i_1 i_2 \ldots i_d} \mathbf{u}_{i_1}^{(1)} \otimes \mathbf{u}_{i_2}^{(2)} \ldots \otimes \mathbf{u}_{i_d}^{(d)}. \tag{5.25}$$



(a) Illustration of the CP format of a tensor of order $d = 3$ in R rank-one tensors

(b) Matrix form of the CP decomposition

Figure 5.6: Polyadic decomposition in $3D$

where $\mathbf{u}_i^{(k)}$, $i = 1, \ldots, n^{(k)}$ are vectors from vector spaces $V^{(k)}$ of dimension $n^{(k)}$ that are, in general, orthonormal. By setting

$$\tilde{\mathbf{G}} = \sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \ldots \ldots \sum_{i_d=1}^{r_d} g_{i_1 i_2 \ldots i_d}$$

we get a tensor called *Tucker core tensor* of order $(r_1, r_2, \ldots, r_d)$ smaller than the original one. In matrix form it writes

$$\mathbf{F} = \tilde{\mathbf{G}} \times_1 U^{(1)} \times_2 U^{(2)} \ldots \times_d U^{(d)}. \tag{5.26}$$

Figure 5.7 illustrates the Tucker decomposition format of a 3-order tensor, where one can consider the core tensor $\tilde{\mathbf{G}}$ as a compressed format of $\mathbf{F}$ since the matrix factor are orthogonal.

The Tucker format is attractive in that the matrix factors are often chosen orthogonal. The advantage being that several operations will be restricted to the single core tensor which is then a compressed form of the initial tensor. The more the compression is significant ($r_k << n^{(k)}, k = 1 : d$) without significative loss, the more efficient the Tucker format.

However, it has the disadvantage of always having a core tensor that is multidimensional and not diagonal. Thus, the algorithmic complexity will still grow exponentially with respect to the dimension d. Hence, the use of Tucker hierarchical formats or Tensor-Train (TT) that try to combine the advantages of CP and Tucker representations is required. In this work, we focus on Tensor-Train formats which can be seen as a special case of Hierarchical Tucker format [6].

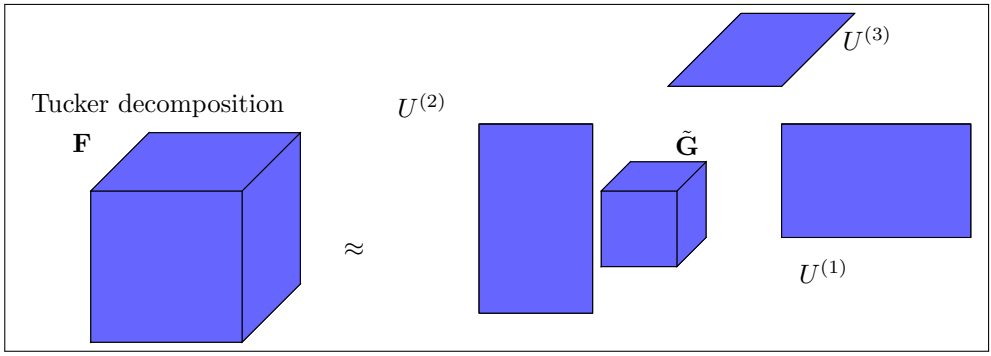


Figure 5.7: Illustration of the Tucker decomposition.

5.2.3 Hierarchical Tucker format

The main idea of the Hierarchical Tucker representation is the way of approximating a numerical tensor $\mathbf{A}$ from a tensor space

$$\mathcal{V} = \otimes_{k=1}^{d} V^{(k)},$$

by a numerical tensor

$$\tilde{\mathbf{A}} \in \tilde{\mathcal{V}} = \otimes_{k=1}^{d} U^{(k)}, \tag{5.27}$$

built via vector subspaces $U^{(k)}$ with minimal rank for each $V^{(k)}$. To make this approximation effective, a dimensional tree is used. This tree allows the dimensionality of the high-order tensor to be reduced by using High Order Singular Value Decomposition (*HOSVD*) and truncature (see [6, 78, 88] for HOSVD).

The binary dimensional tree T is the most used. One considers that the dimension set $D = \{1, ..., d\}$ is the root of the tree, it is at level 0. Each node of T is a non-empty subset of D. Any node t of T such that $\#t = 1$ is called *leaf* of T and the set of all leaves is denoted $L(T)$. The notation $I(T) = T \setminus L(T)$ represents the set of nodes t such that there are two children t_1 and t_2, each non-empty with $t = t_1 \cup t_2$ and $t_1 \cap t_2 = \emptyset$. To each node $t \in T$ one defines a maximal rank r such that at each level l, the t-rank r_t is smaller than r. The t-rank r_t is defined in (5.7). If a node t is at level l, its children $\{t_1, t_2\}$ are at level $l + 1$.

Hence, the set of all hierarchical Tucker tensors is given by:

$$\mathcal{H}_r^T(\mathcal{V}) = \{\mathbf{A} \in \mathcal{V} : r_t(\mathbf{A}) \leq r, \quad \forall t \in T\}. \tag{5.28}$$

Any element of $\mathcal{H}_r^T(\mathcal{V})$ can be written with the form:

$$\mathbf{A} = \sum_{i=1}^{r_D} \sum_{j=1}^{r_D} \alpha_{ij}^{D} \mathbf{u}_i^{D_1} \otimes \mathbf{u}_j^{D_2}, \; D = D_1 \cup D_2, \; D_1 \cap D_2 = \emptyset, \; \alpha_{ij}^{D} \in \mathbb{R}, \tag{5.29}$$

and

$$\mathbf{u}_k^{(t)} = \sum_{i=1}^{r_t} \sum_{j=1}^{r_t} \alpha_{ijk}^{(t)} \mathbf{u}_i^{(t_1)} \otimes \mathbf{u}_j^{(t_2)}, \; t = t_1 \cup t_2, \; t_1 \cap t_2 = \emptyset, \; \alpha_{ijk}^{(t)} \in \mathbb{R}, \; \forall t \in I(T), \tag{5.30}$$

where $I(T) = T \setminus D$. A tensor is determined in Hierarchical Tucker format by defining the *transfer tensor* $(\alpha^{(t)})_{t \in I(T)}$ and its *space vector basis* $(\mathbf{u}_i^{(t)})_{t \in L(T), i \in \{1,\dots,r_t\}}$.

One way of determining the bases

$$B^{(t)} = [\mathbf{u}_i^{(t)}] \in U^{(t)}, i = 1, 2, \dots, r_t \tag{5.31}$$

and the transfer tensors $\alpha_{ijk}^{(t)}$ refers to the so called High-order Singular Value Decomposition that we present below.

High Order Singular Value Decomposition (HOSVD)

The matricization map (5.7) allows us to define a matrix that correspond to the tensor $(\mathbf{A})$ at each node of the tree in mode t, denoted $M_t(\mathbf{A})$ (where t is a subset of the dimension set D). Considering the left singular vectors $\mathbf{u}_i^{(t)}$ of $M_t(\mathbf{A})$, one forms an orthonormal basis

$$B^{(t)} = [\mathbf{u}_1^{(t)}, \mathbf{u}_2^{(t)}, \dots, \mathbf{u}_{r_t}^{(t)}]$$

such that

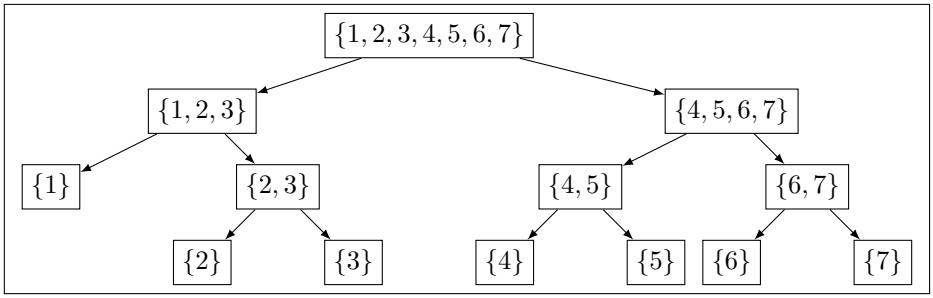
$$U^{(t)} = span\{\mathbf{u}_i^{(t)}, 1 \leq i \leq r_t\}$$

for all $t \in T$. Such bases at each vertex of the tree form the so-called HOSVD bases (see [6, p.339]). However, if one should store all the entries of these bases, it would require huge storage. Instead, the Hierarchical representation uses some recursive frames that allow to express the bases at vertex $t \in T$ by means of its sons bases $B^{(t_1)}$ and $B^{(t_2)}$. In this case, only the bases of leaf nodes ($t \in L(T)$ is a singleton) have to be stored explicitly while bases at higher levels are implicitly formed via kronecker product and the so-called transfert tensors (see [6]). The coefficients $\alpha_{ijk}^{(t)}$ may be built by the left singular values of the matrix $M_t(\mathbf{A})$ or by eigenvalues of

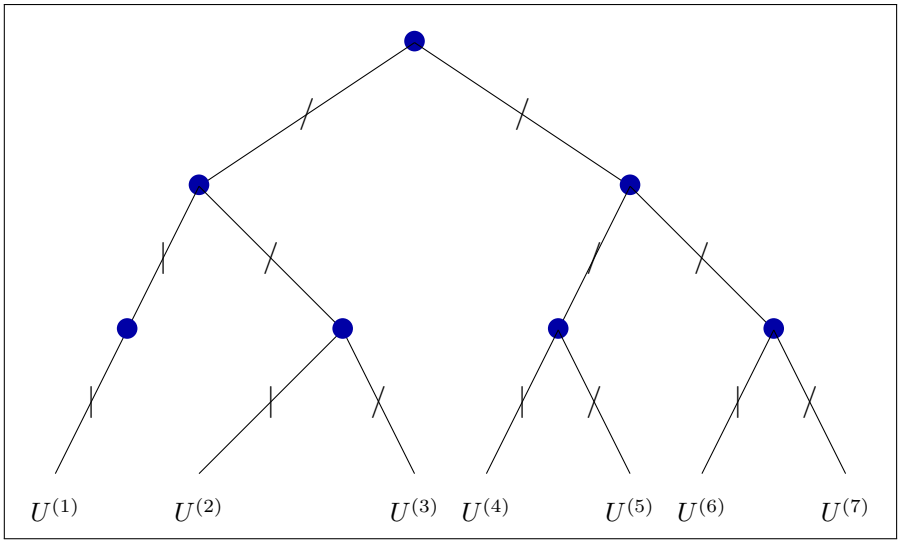
$$M_t(\mathbf{A}) M_t(\mathbf{A})^T$$

with some recursive truncations. The procedure is repeated for each subset $t \subset D$ as described in [6, 11.3.3]. This procedure one of the High Order Singular Value Decomposition *HOSVD* that is a generalizing the matrix SVD. For more details, we refer to [6, Chap. 11] for this construction while other HOSVD in other formats are presented in, for example [78] and referencies therein.

Figure 5.8 illustrates a hierarchical tensor tree Figure 5.8a of a 7-dimensional tensor $\mathbf{A}$ and its representation in hierarchal Figure 5.8b format following the concepts in 5.1.2. The first node at level 0 is the root of the tree with $D = \{1, \dots, 7\}$ and each next level is formad by two disjoind subset of the previous. In Figure 5.8b the $\bullet$ indicates a contracted product between two tensors and the number of bared segment indicates the resulting dimension of the result tensor. One can see that (by observing Figure 5.8b), at the root tensor the result tensor is a matrix (two dimension) while each non leaf and non root node represents a 3-dimensional tensor. The leaves represent the unidimensional subspaces $U^{(k)}$, $k = 1 : d$ from where are constructed unidimensional bases $B^{(k)}$ defined at (5.31).



(a) Dimension tree:$\mathbf{A}$ ($d = 7$).



(b) Hierarchical Tensor decomposition $\mathbf{A}$ ($d = 7$)

Figure 5.8: Compression hierarchical tree ($d = 7$)

5.3 Tensor-Train decomposition and low-rank Approximation

5.3.1 Tensor-Train(TT) format and TT-decomposition

In this section and in the remainder, A^T and $\mathbf{v}^T$ indicate, respectively, the transpose of a matrix A and the transpose of a vector $\mathbf{v}$.

A Tensor-Train format of a multidimensional array can be considered as a particular case of its Hierarchical Tensor-format. In this case, each node of the binary dimension tree is formed by a singleton and its complementary (see for example [6, pp. 374-375] for more details).

The main motivations for using the TT-format are both the possibility of representing exactly a high-order tensor with few parameters or the possibility of approx-

imating it by a low-rank tensor with a given accuracy by fewer parameters [91]. In addition, efficient algorithms have been addressed to compute optimal ranks r_k enabling a given high-order tensor to be represented in the TT-format with a required accuracy ϵ. Further, reshaping large-scale matrices and vectors into high-order tensors and approximating them in TT format allows certain algebraic operations to be performed such as addition and matrix-vector product in logarithmic time complexity [91].

The TT-representation of a tensorized vector or matrix $\mathbf{A}$ can be considered as a low-rank representation performed successively on its matricizations $\mathbf{A}^{(k)}, k = 1 : d$. These matricizations often lead to linear dependence among rows or columns that may be removed by standard transformation of rows and columns. Thus, the TT decomposition of large vectors and large-scale matrices uses a recursive Kronecker-product representation with use of *common bases* [92].

To represent a tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ in TT format, we have to determine l components[3] $(l \geq d)$ $U^{(k)},\ k = 1, ..., l$ such that

$$\mathbf{A}(i_1, i_2, \ldots, i_l) \approx \sum_{\alpha_0, \alpha_1, \ldots, \alpha_l} \mathbf{U}^{(1)}(\alpha_0, i_1, \alpha_1)\mathbf{U}^{(2)}(\alpha_1, i_2, \alpha_2) \ldots \mathbf{U}^{(l)}(\alpha_{l-1}, i_l, \alpha_l), \tag{5.32}$$

where the components $\mathbf{U}^{(k)},\ 1 \leq k \leq l$ are called *core tensors*. In general, each $\mathbf{U}^{(k)}$ is a 3-dimensional array for large vectors.

For example, consider a vector

$$\mathbf{x} \in \mathbb{R}^{n^{(1)} . n^{(2)} \ldots . n^{(d)}}.$$

This vector can be reshaped in an array $\mathbf{x} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$, where $\#I_k = n^{(k)},\ \forall k \in \{1, 2, \ldots, l\}$. To convert this vector in the form (5.32), each core will depend on one initial index $i_k \in I_k$ and on two auxiliary index variables $\alpha_{k-1} \in K_{k-1}, \alpha_k \in K_k, \forall\ k \in K_k = \{1, 2, \ldots, r_k\}, k = 1 : l$. The numbers r_k are called the TT-rank of the vector and $r_1 = r_l = 1$ by definition. We may refer to this as a *TT-vector* and using Kronecker product of mode-2 fibers (columns), we may write

$$\mathbf{x} \approx \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \ldots \sum_{\alpha_{l-1}=1}^{r_{l-1}} = \mathbf{U}^{(1)}(1, i_1, \alpha_1) \otimes \mathbf{U}^{(2)}(\alpha_1, i_2, \alpha_2) \otimes \ldots \otimes \mathbf{U}^{(l)}(\alpha_{l-1}, i_l, 1).$$

For a large-scale matrix $A \in \mathbb{R}^{m \times n}$, the TT-representation is obtained in the same way by extension of (5.32) where each core is now a 4-dimensional array. The matrix A can be reshaped as an array

$$\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_{d_1} \times J_1 \times J_2 \times \ldots \times J_{d_2}},\ \ d_1, d_2 \in \mathbb{N}$$

called *TT-matrix*, where

$$m = \prod_{k=1}^{d_1} \#I_k$$

3. The new dimension l is used as an artificial dimension and has to be distinguished from the real dimension d. When $l = d$, then the real dimension of the problem is used.

and

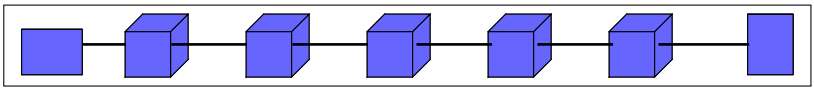
$$n = \prod_{k=1}^{d_2} \#J_k.$$

Thus, one can approximate its TT-representation using Kronecker product of mode-2 slices (matrices) and write

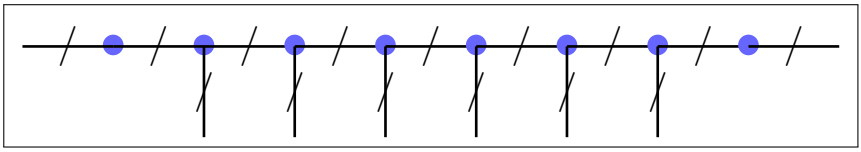
$$\mathbf{A} \approx$$

$$\sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \dots \sum_{\alpha_{l-1}=1}^{r_{l-1}} \mathbf{U}^{(1)}(1, i_1, j_1, \alpha_1) \otimes \mathbf{U}^{(2)}(\alpha_1, i_2, j_2, \alpha_2) \otimes \dots \otimes \mathbf{U}^{(l)}(\alpha_{l-1}, i_l, j_l, 1).$$

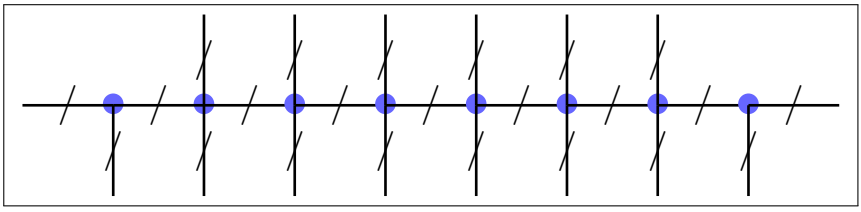
Figure 5.9 visualizes the TT-format where for TT-vectors, the boundary conditions give rise to matrices on extremes and 3-order tensors any where else, linked by mode-1 contracted product. The form of a train is shown by Figure 5.9a while the equivalent but more general form is shown by Figure 5.9b. Figure 5.9c visualizes the TT-representation of large matrices where we have 3-dimensional arrays on extremes and 4-dimensional arrays anywhere else.



(a) Tensor-Train format of 7^{th}-order TT-vector



(b) Tensor-Train format of 7^{th}-order TT-vector.



(c) Tensor- Train format of 7^{th}-order TT-matrix represented

Figure 5.9: Tensor-Train format of 7^{th}-order. The form of a train is shown by Figure 5.9a while the equivalent but more general form is shown by Figure 5.9b. Figure 5.9c visualizes the TT-representation of large matrices where we have 3-dimensional on extremes and 4-dimensional arrays anywhere else.

Let us now show how these cores are built. First, recall that when some variables separability is possible, any matrix $A \in \mathbb{R}^{m \times n}$ can be written as a sum of Kronecker

product of certain vectors:

$$A = \sum_{j=1}^{r} u_j \otimes v_j = \sum_{j=1}^{r} u_j v_j^T,$$

where r is the rank of the matrix A [92]. This allows to write a matrix A in the form of dyadic (skeleton) decomposition: $A = UV^T$. In other words, the matrix A whose entries can be denoted $A(i,j),\ i = 1:m,\ j = 1:n$, can be considered as a Kronecker product of two matrices $G^{(1)} \in \mathbb{R}^{m^{(1)} \times n^{(1)}}$ and $G^{(2)} \in \mathbb{R}^{m^{(2)} \times n^{(2)}}$ such that $(m,n) = (m^{(1)}.m^{(2)}, n^{(1)}.n^{(2)})$. This means we can write

$$A = G^{(1)} \otimes G^{(2)} = [G^{(1)}(i_1, j_1)G^{(2)}]\ \ i_1 = 1 : m^{(1)},\ j_1 = 1 : n^{(1)}. \tag{5.33}$$

Thus, the decomposition relies on a certain splitting of spatial indices and can be done recursively. To understand this split, let us consider row and column indices of A as multi-indices:

$$(i,j) = (i_1 i_2, j_1 j_2), \ \text{ and } \ A(i,j) = A(i_1 i_2, j_1 j_2).$$

Observe that the indices i and j from the Kronecker product $A = G^{(1)} \otimes G^{(2)}$ are not separated but are products of mode-indices [92]. The idea of variables separation is to reshape the matrix A in a new matrix $\tilde{A}$ with special multi-indices that admit separation. This means, we should reshape the matrix A in the form

$$\tilde{A}(i_1 j_1, i_2 j_2) \approx A(i_1 i_2, j_1 j_2)\ \ i_1 = 1 : m^{(1)},\ i_2 = 1 : m^{(2)},\ j_1 = 1 : n^{(1)},\ j_2 = 1 : n^{(2)}, \tag{5.34}$$

such that $\tilde{A}(i_1 j_1, i_2 j_2)$ is a *matrix of low rank*. Since most matrices are not represented by a single Kronecker product, we may have a sum of Kronecker products that writes

$$\tilde{A} = \sum_{\alpha=1}^{r} G_\alpha^{(1)} \otimes G_\alpha^{(2)}, \tag{5.35}$$

where $G_\alpha^{(k)}$ is a matrix. The natural integer r is known as the *Kronecker-rank of the representation* while it is called the *Kronecker-rank of the matrix* $\tilde{A}$ when it is the minimal possible for all the representations of the form (5.35) (see [92]).

The right-hand side of the equation (5.35) allows to write the matrix A with

$$r(m^{(1)}n^{(1)} + m^{(2)}n^{(2)})$$

entries that can be significantly smaller than

$$mn = m^{(1)}n^{(1)}m^{(2)}n^{(2)},$$

for small r. Thus, the number of parameters to represent the matrix A has been reduced. In order to futher compress the representation, we may increase the number of Kronecker factors while decreasing the size of each matrix factor. That is,

$$\tilde{A} = \sum_{\alpha=1}^{r} \otimes_{k=1}^{l} G_\alpha^{(k)}, \tag{5.36}$$

where $G_\alpha^{(k)}$ is a parameter dependent matrix. With this formulation, the size of each $G_\alpha^{(k)}$ is decreased but $l,\ l \geq d$, factors are used. This means that we have enlarged the dimension d while we have created small blocks that form the core tensor. This core tensor is expected to have small rank r. Consider, for simplicity, that the matrix A is of order n. One can observe that the main advantage depends on the rate of increase of r when n decreases. If r increases slowly while n has significantly decreased, we get an important reduction of parameters. Instead of manipulating n^2 entries od A one may manipulate N_p^{TT} parameters where

$$N_p^{TT} << n^2.$$

In particular, when the order of A is a power of 2, that is, $n = 2^d$, we may enlarge the dimension to the maximal tensor dimension d while each $G_\alpha^{(k)}$ has now 2×2 blocks. In this case, the representation is called *Quantized Tensor-Train (QTT) representation.* Such a representation allows to reduce significantly the number of parameters in the tensor. The number of parameters is then given (see [80]) by

$$N_p^{QTT} = 4rlog_2 n. \tag{5.37}$$

Consider now we need to approximate a d-dimensional tensor of size $[n, n, \ldots, n]$ and n^d entries, by a new tensor with few parameters, compared to n^d. For this purpose, a TT-vector $\mathbf{a}$ and a TT-matrix $\mathbf{A}$ have to be approximated by new tensors $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{A}}$ of respective forms

$$\tilde{\mathbf{a}} = \mathbf{G}^{(1)}(j_1)\mathbf{G}^{(2)}(j_2)\ldots\mathbf{G}^{(d)}(j_d), \tag{5.38}$$

and

$$\tilde{\mathbf{A}} = \mathbf{G}^{(1)}(i_1, j_1)\mathbf{G}^{(2)}(i_2, j_2)\ldots\mathbf{G}^{(d)}(i_d, j_d), \tag{5.39}$$

where each

$$\mathbf{G}^{(k)}(j_k) = \mathbf{U}^{(k)}_{r_{k-1}, j_k, r_k} \quad \text{and} \quad \mathbf{G}^{(k)}(i_k, j_k) = \mathbf{U}^{(k)}_{r_{k-1}, i_k, j_k, r_k}$$

respectively. Each factor of this notation is a kind of parameter-dependent block matrix with $r_{k-1} \times r_k$ blocks where the d-tuple $(r_0, r_1, \ldots, r_d)$ is called *TT-ranks* and the relations $r_0 = r_d = 1$ are imposed as boundary conditions. The maximal rank

$$r = \max\{r_k\},\ k = 1 : d-1$$

is expected to be *small* while, given a threshold ϵ $(0 < \epsilon < 1)$ the approximation verifies

$$\|\mathbf{a} - \tilde{\mathbf{a}}\|_F < \epsilon,$$

respectively,

$$\|\mathbf{A} - \tilde{\mathbf{A}}\|_F < \epsilon$$

where $\|.\|_F$ denotes the Frobenius norm. In general, the approximation is done via k-matricization 5.9 in each mode. An upper bound on the rank r_k is given by the theorem below from [80, p. 2298].

Theorem 5.7

If for each unfolding matrix of the form (5.9) *of a d-dimensional tensor* $\mathbf{A}$ *such that* $rank(A^{(k)}) = r_k$, *then there is a decomposition* (5.38), *respectively* (5.39)

with TT-ranks not higher than r_k where r_k is the rank of the k-matricization, $A^{(k)}$, of the tensor $\mathbf{A}$.

hence, the decomposition may be performed via singular value or QR decompositions.

Proof. See [80, p. 2298] □

Now it may be important to know how better is the approximation in low-rank TT-tensor format. The following theorem and corollaries prensent the expected accuracy when approximating a low-rank Tensor-Train with a given accuracy using singular value decomposition.

Theorem 5.8 (*See [80]*)

Suppose the unfolding matrix $A^{(k)}$ of the tensor $\mathbf{A}$ verifies

$$A^{(k)} = B^{(k)} + E^{(k)}, \quad \text{with } rank(A^{(k)}) = r_k, \text{ and } \|E^{(k)}\|_F = \epsilon_k, \tag{5.40}$$

where $k = 1, 2, \ldots, d-1$. Then the TT-SVD computes a tensor in TT format with TT-ranks r_k and

$$\|\mathbf{A} - \tilde{\mathbf{B}}\|_F \leq \sqrt{\sum_{k=1}^{d-1} \epsilon_k^2}. \tag{5.41}$$

Proof. See [80, p. 2299] □

Two corollaries (proved in [80] and in [93]) provide information on how better is the TT-approximation compared to the CP approximation and the best approximation via SVD.

Corollary 5.9

If a tensor $\mathbf{A}$ admits a canonical approximation with R terms and accuracy ϵ, then there exists a TT-approximation with TT-ranks $r_k \leq R$ and accuracy $\sqrt{d-1}\epsilon$.

Corollary 5.10

Given a tensor $\mathbf{A}$ and TT-ranks bounds r_k, the best approximation to $\mathbf{A}$ in the Frobenius norm with TT-ranks, denoted by $\mathbf{A}_{best}$, always exist and the TT-approximation $\mathbf{B}$ computed by TT-SVD algorithm is quasi-optimal and we have that

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \sqrt{d-1}\|\mathbf{A} - \mathbf{A}_{best}\|_F. \tag{5.42}$$

This last corollary compares the TT-approximation computed by SVD with the best approximation in Frobenius norm.

Now, we need to link the TT-Tensor operations to known matrix operations. The lemma below, that is proved in [92] explains that a matrix that can be written as Kronecker product can be considered as a block matrix. This allows to link the matrix operations with TT-tensor operations since one may be manipalating the blocks as matrices.

Definition 5.11

Given a decomposition of the form (5.35) and, suppose that A is of order $m \times n$, with $m = m^{(1)}m^{(2)}$ and $n = n^{(1)}n^{(2)}$ such that $G_\alpha^{(1)}$ and $G_\alpha^{(2)}$ are of order $m^{(1)} \times$

$n^{(1)}$ *and* $m^{(2)} \times n^{(2)}$*, respectively. Then* A *can be considered as a block matrix:*

$$A = [A_{i_1 j_1}], \quad 1 \leq i_1 \leq m^{(1)}, 1 \leq j_1 \leq n^{(1)}, \tag{5.43}$$

refered to as the core matrix of A*.*

This definition will serve in the reminder to manipulate the small blocs that form the core matrix of differential operator we use in MIRP.

Let us adress the possibility to solve more efficiently the linear system (4.9) using TT-Tensor format. For this purpose, we focus on Quintized TT (QTT) representations of tridiagonal Toeplitz-like matrices, and especially the negative Laplace operator in a d-dimensional space. The *Quantized Tensor Train (QTT)*format is a TT-representation applied to a vector or a matrix after introduction of virtual dimensions by reshaping the vector or the matrix to smaller mode size (in general $n^{(k)}$ is reduced to 2, $\forall 1 \leq k \leq l$).

From the definition above, given an array $\mathbf{A}$ with size

$$(m^{(1)} \times \ldots \times m^{(d)}) \times (n^{(1)} \times \ldots \times n^{(d)}),$$

its TT-representation given by

$$\begin{aligned} \mathbf{A}_{\substack{i_1,\ldots,i_d \\ j_1,\ldots,j_d}} \quad = \quad & \sum_{\alpha_1}^{r_1} \ldots \sum_{\alpha_{d-1}}^{r_{d-1}} U^{(1)}(\alpha_0, i_1, j_1, \alpha_1) U^{(2)}(\alpha_1, i_2, j_2, \alpha_2) \\ & \ldots U^{(d)}(\alpha_{d-1}, i_d, j_d, \alpha_d), \end{aligned} \tag{5.44}$$

can be interpreted as contracted product of four-dimensional arrays. In this case, each four-dimensional array $U^{(k)}$, $1 \leq k \leq d$ is a block matrix that is considered as the *TT-core* or *core matrix* of $\mathbf{A}$. To be explicit, let us consider one TT-core U as a 4 dimensional array of size (p, m, n, q).

Definition 5.12

The TT-core U *is said to have TT-core ranks or TT-ranks* (p, q) *and mode size* (m, n) *if it is formed by* pq *blocks of size* (m, n) *each. That is*

$$U(\alpha_1, i, j, \alpha_2) = (U_{\alpha_1,\alpha_2})_{i,j}, \quad \alpha = 1 : p,\ \alpha_2 = 1 : q,\ i = 1 : m,\ j = 1 : n.$$

Then we can write

$$U = \begin{bmatrix} U_{11} & \ldots & U_{1q} \\ \vdots & \ddots & \vdots \\ U_{p1} & \ldots & U_{pq} \end{bmatrix}. \tag{5.45}$$

To perform operations with these core matrices, a specific product called *rank core product* and denoted by $\bowtie$ is defined below (see [81, p.745]).

Definition 5.13 (*Rank-core product*)

Consider core matrices $U^{(1)}$ *and* $U^{(2)}$ *with respective core-ranks* $r_0 \times r_1$ *and* $r_1 \times r_2$ *composed by blocs* $U^{(1)}_{\alpha_0\alpha_1}$ *for* $U^{(1)}$ *and* $U^{(2)}_{\alpha_1\alpha_2}$*,for* $U^{(2)}$ $\quad 1 \leq \alpha_k \leq r_k,\ 0 \leq k \leq 2$*, where each block is of mode size* $m^{(1)} \times n^{(1)}$ *in the core* $U^{(1)}$ *and* $m^{(2)} \times n^{(2)}$ *in the core* $U^{(2)}$*. The rank-core product of* $U^{(1)}$ *and* $U^{(2)}$ *is a core matrix* $U = U^{(1)} \bowtie$

$U^{(2)}$ *of core rank* $r_0 \times r_2$ *and blocks of mode sizes* $m^{(1)}n^{(1)} \times m^{(2)}n^{(2)}$*:*

$$U_{\alpha_0\alpha_2} = \sum_{\alpha_1}^{r_1} U^{(1)}_{\alpha_0\alpha_1} \otimes U^{(2)}_{\alpha_1\alpha_2},\ 1 \le \alpha_0 \le r_0,\ 1 \le \alpha_2 \le r_2.$$

Observe that, the rank-core product can be considered as a usual matrix-by-matrix product where matrix elements are replaced by blocks and the usual multiplication is replaced by the Kronecker product

$$\begin{bmatrix} U^{(1)}_{11} & U^{(1)}_{12} \\ U^{(1)}_{21} & U^{(1)}_{22} \end{bmatrix} \bowtie \begin{bmatrix} U^{(2)}_{11} & U^{(2)}_{12} \\ U^{(2)}_{21} & U^{(2)}_{22} \end{bmatrix}$$
$$= \begin{bmatrix} U^{(1)}_{11} \otimes U^{(2)}_{11} + U^{(1)}_{12} \otimes U^{(2)}_{21} & U^{(1)}_{11} \otimes U^{(2)}_{12} + U^{(1)}_{12} \otimes U^{(2)}_{22} \\ U^{(1)}_{21} \otimes U^{(2)}_{11} + U^{(1)}_{22} \otimes U^{(2)}_{21} & U^{(1)}_{21} \otimes U^{(2)}_{12} + U^{(1)}_{22} \otimes U^{(2)}_{22} \end{bmatrix}.$$

With this notation, we can write the relation (5.39) by

$$\tilde{\mathbf{A}} = \mathbf{U}^{(1)} \bowtie \mathbf{U}^{(2)} \bowtie \ldots \bowtie \mathbf{U}^{(d)}. \tag{5.46}$$

The rank-core product verifies the properties below. For $l \in \mathbb{N},\ l \ge 2$ and tensors

$$\mathbf{A} = \mathbf{U}^{(1)} \bowtie \mathbf{U}^{(2)} \bowtie \ldots \bowtie \mathbf{U}^{(l)},\ \ \mathbf{B} = \mathbf{V}^{(1)} \bowtie \mathbf{V}^{(2)} \bowtie \ldots \bowtie \mathbf{V}^{(l)},$$

with appropriate core ranks and mode sizes, we have:

$$\alpha\mathbf{A} + \beta\mathbf{B} = \begin{bmatrix} U^{(1)} & V^{(1)} \end{bmatrix} \bowtie \begin{bmatrix} U^{(2)} & \\ & V^{(2)} \end{bmatrix} \bowtie \ldots \bowtie \begin{bmatrix} U^{(d-1)} & \\ & V^{(d-1)} \end{bmatrix} \bowtie \begin{bmatrix} \alpha U^{(d)} \\ \beta V^{(d)} \end{bmatrix}.$$

From the Kronecker product and multilinearity it can be verified that

$$\begin{bmatrix} \alpha_1 U^{(1)} & \beta_1 U^{(1)} \\ \alpha_1 U^{(1)} & \beta_1 U^{(1)} \end{bmatrix} \bowtie \begin{bmatrix} \alpha_2 U^{(2)} & \alpha_2 U^{(2)} \\ \beta_2 U^{(2)} & \beta_2 U^{(2)} \end{bmatrix}$$
$$= \left(\begin{bmatrix} U^{(1)} \\ V^{(1)} \end{bmatrix} \bowtie \begin{bmatrix} \alpha_1 & \beta_1 \end{bmatrix} \right) \bowtie \left(\begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \bowtie \begin{bmatrix} U^{(2)} & V^{(2)} \end{bmatrix} \right)$$
$$= (\alpha_1\alpha_2 + \beta_1\beta_2) \begin{bmatrix} U^{(1)} \\ V^{(1)} \end{bmatrix} \bowtie \begin{bmatrix} U^{(2)} & V^{(2)} \end{bmatrix}.$$

This property allows to detect linear dependace of rows or columns in the core matrices and thus reveals the true TT-rank of the TT-representation. In particular, for QTT-representation, the blocks below are used to desctibe and manipulate QTT structures of tridiagonal Toeplitz-like matrices (see [81, p.747]):

$$\begin{aligned} E &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad F = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad E_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad E_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \\ G &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad L = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{5.47}$$

From these small matrices and by noting

$$A^{\bowtie k} = \overbrace{A \bowtie A \bowtie \ldots \bowtie A}^{k\,factors},$$

the proposition below allows a QTT structure of any tridiagonal Toeplitz matrix to be represented in QTT format.

Proposition 5.14

Assume $l \geq 2$ and α, β, γ are real numbers. Then the matrix

$$A^{(l)} = \begin{pmatrix} \alpha & \beta & & & \\ \gamma & \alpha & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \alpha & \beta \\ & & & \gamma & \alpha \end{pmatrix},$$

of size $2^l \times 2^l$ has the following QTT representation of TT-ranks $3, 3, \ldots, 3$.

$$A^{(l)} = \begin{pmatrix} E & F^T & F \end{pmatrix} \bowtie \begin{pmatrix} E & F^T & F \\ & F & \\ & & F^T \end{pmatrix}^{\bowtie(l-2)} \bowtie \begin{pmatrix} \alpha E + \beta F + \gamma F^T \\ \gamma F \\ \beta F^T \end{pmatrix}. \quad (5.48)$$

Denote that, the TT-rank $(3, 3, \ldots,)$ means each core matrix has three small blocks. Now, we can apply this proposition to unidimensional and multidimensional Laplace operators following the developments in [81]. Consider $d > 2$ and $l_k \in \mathbb{N}$, $l_k > 1$ for $1 \leq k \leq d$. The unidimensional matrix $\Delta^{(l_k)}, 1 \leq k \leq d$, with size $2^{l_k} \times 2^{l_k}$, is given by

$$\Delta_{DD}^{(l_k)} = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \text{ or } \Delta_{DN}^{(l_k)} = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix},$$

where the $\Delta_{DD}^{(l_k)}$ is the unidimensional Laplace operator with Dirichlet boundary condition while $\Delta_{DN}^{(l_k)}$ is the same operator but with mixed Dirichlet-Neumann boundary conditions. The equality

$$\Delta_{DN}^{(l_k)} = \Delta_{DD}^{(l_k)} - I_*^{\otimes d} \quad (5.49)$$

where

$$I_* = \begin{pmatrix} & & \\ & & \\ & & 1 \end{pmatrix}$$

and $I_*^{\otimes d} = \overbrace{I \otimes I \otimes \ldots \otimes I}^{d\ factors}$ allows to apply results from one of the two operators to the other. Although we are interested in the d-dimensional Laplace operator expressed by

$$\Delta^{(l_1,\ldots,l_d)} = a_1 \Delta^{(l_1)} \otimes M^{(l_2)} \otimes \ldots \otimes M^{(l_d)} + \ldots + M^{(l_1)} \otimes \ldots \otimes M^{(l_{d-1})} \otimes a_d \Delta^{(l_d)}, \quad (5.50)$$

we need to express first the QTT structure of unidimensional operator and generalize after. For finite difference methods, the matrix $M^{(l_k)} = I$ is the identity while for finite element methods, $M^{(l_k)}$ can be given by other tridiagonal matrices (see [81]).

The QTT representation of the unidimensional Laplace operators are obtained by applying the Proposition 5.14 with $\alpha = 2, \beta = -1$ and $\gamma = -1$:

Corollary 5.15

For $l \geq 3$ we have

$$\Delta_{DD}^{(l)} = \begin{pmatrix} E & F^T & F \end{pmatrix} \bowtie \begin{pmatrix} E & F^T & F \\ & F & \\ & & F^T \end{pmatrix}^{\bowtie(l-2)} \bowtie \begin{pmatrix} 2E - F - F^T \\ -F \\ -F^T \end{pmatrix} \tag{5.51}$$

and using also (5.49)

$$\Delta_{DN}^{(l)} = \begin{pmatrix} E & F^T & F & E_2 \end{pmatrix} \bowtie \begin{pmatrix} E & F^T & F & \\ & F & & \\ & & F^T & \\ & & & E_2 \end{pmatrix}^{\bowtie(l-2)} \bowtie \begin{pmatrix} 2E - F - F^T \\ -F \\ -F^T \\ -E_2 \end{pmatrix}. \tag{5.52}$$

From these representations, one can observe that following definition 5.12, the TT-ranks of $\Delta_{DD}^{(l)}$ and $\Delta_{DN}^{(l)}$ is $3, 3, \ldots, 3$ and $4, 4, \ldots, 4$. respectively. We have in each core matrix 3 and 4 blocks respectively.

The multidimensional Laplace operator writes in the general form

$$\begin{array}{rcr} \mathbf{L}^{(d)} & = & B^{(1)} \otimes M^{(2)} \otimes M^{(3)} \otimes \ldots \otimes M^{(d-2)} \otimes M^{(d-1)} \otimes M^{(d)} \\ & + & L^{(1)} \otimes B^{(2)} \otimes M^{(3)} \otimes \ldots \otimes M^{(d-2)} \otimes M^{(d-1)} \otimes M^{(d)} \\ & + & \ldots \otimes L^{(1)} \otimes L^{(2)} \otimes L^{(3)} \otimes \ldots \otimes L^{(d-2)} \otimes B^{(d-1)} \otimes M^{(d)} \\ & + & L^{(1)} \otimes L^{(2)} \otimes L^{(3)} \otimes \ldots \otimes L^{(d-2)} \otimes L^{(d-1)} \otimes B^{(d)} \end{array} \tag{5.53}$$

where $L^{(k)}, M^{(k)}$ and $B^{(k)}$ are matrices of size $m^{(k)} \times n^{(k)}, 1 \leq k \leq d$. On can verify (see [81])that they can be represented in QTT format with TT-core ranks $(2, \ldots, 2)$ in terms of blocks $L^{(k)}, M^{(k)}$ and $B^{(k)}$. That is,

$$\mathbf{L}^{(d)} = \begin{pmatrix} L^{(1)} & B^{(1)} \end{pmatrix} \bowtie \begin{pmatrix} L^{(2)} & B^{(2)} \\ & R^{(2)} \end{pmatrix} \bowtie \ldots \bowtie \begin{pmatrix} L^{(d-1)} & B^{(d-1)} \\ & M^{(d-1)} \end{pmatrix} \bowtie \begin{pmatrix} B^{(d)} \\ M^{(d)} \end{pmatrix}. \tag{5.54}$$

In our case, $L^{(k)} = M^{(k)} = I^{(k)}$ and $B^{(k)} = \Delta_{DD}^{(l_k)}$.

Observe that, since Kronecker product is successively used, small quantities may be multiplied. This results in very small quantities that may lead to numerical instabilities. Thus, truncation is required when working with TT-tensor format. This is addressed in the following subsection.

5.3.2 Rounding in TT format

We aim to analyze rounding errors that may arise when either converting a general tensor in TT-tensor format or when truncating a TT-tensor after certain operations.

Converting a general tensor into a TT-tensor using, for example TT-SVD, is already expensive for high-dimensional tensors. However, if the tensor is given in a certain structured format, the complexity of the task may be significantly reduced. Consider a tensor that is already in TT format but with TT-ranks that are suboptimal. Such suboptimal ranks are often greater than the real ranks and may grow whenever some linear operations (such as addition, tensor-vector product or mode-k products) are performed.

One way of avoiding the TT-ranks growth is the use of some compression techniques and certain truncature to reduce these ranks while maintaining the accuracy as good as possible. Many iterative tensorial operations tend to increase the rank of the result tensor. When these operations are repeated many times, the rank will continue to rise and may explode. For this reason, it is judicious to replace after each operation a result $\mathbf{A}_s$ of rank s sufficiently large by its approximation $\mathbf{B}_r$ of rank r relatively smaller.

The truncation operator $T_{r,s}$ is defined either by fixing the destination rank r or by fixing a threshold $\epsilon > 0$ such that $||\mathbf{A}_s - \mathbf{B}_r|| < \epsilon$ and $r < s$. Thus,

$$\mathbf{B}_r = T_{r,s}(\mathbf{A}) \Leftrightarrow ||\mathbf{A}_s - \mathbf{B}_r|| < \epsilon, \quad r < s, \tag{5.55}$$

within an appropriate norm. When the destination rank r is fixed we may denote the truncation operator simply T_r. A result in [6, p.354] (Theorem 11.56) proves the existence of a best approximation for problems in hierarchical tensor format and thus in TT format. When this approximation ($T_{r,s}$) uses the HOSVD, that is ($s - r$) smallest eigenvalues and their associated eigenvectors are neglected in each k-matricization, the theorem 11.64 in [6, 362] shows that the approximation verifies

$$||\mathbf{A} - \mathbf{B}_r|| \leq \sqrt{2d-3}||\mathbf{A} - \mathbf{B}_{best}||. \tag{5.56}$$

where d is the dimension of these tensors, s is the maximal rank of the actual tensor $\mathbf{A}$ and r is the destination rank, that is the fixed maximal rank of the tensor $\mathbf{B}$ approximating $\mathbf{A}$.

In this thesis we present another technique proposed by Oseledets for TT-decomposition [80]. This technique uses successive QR and SVD decompositions. Consider a tensor in the form (5.39) with TT-ranks (r_k), $k = 1, \ldots, d-1$. The aim is to estimate the optimal smaller rank $(r_k^{'})$, $r_k^{'} \leq r_k$, $k = 1, \ldots, d-1$ while the accuracy is maintained as high as possible. This *truncation* operation should allow a low-parametric representation of the tensor.

Consider the tensor $\mathbf{A} = \mathbf{G}^{(1)}(j_1)\mathbf{G}^{(2)}(j_2)\ldots\mathbf{G}^{(d)}(j_d)$ described in (5.38). To compress this tensor, we may first matricize it in, for example, mode-one and apply its dyadic decomposition:

$$A^{(1)} = UV^T, \tag{5.57}$$

where

$$U(j_1, \alpha_1) = \mathbf{G}^{(1)}(j_1, \alpha_1) \tag{5.58}$$

and

$$V(j_2, \ldots, j_d, \alpha_1) = \mathbf{G}^{(2)}(\alpha_1, j_2)\mathbf{G}^{(3)}(j_3)\ldots\mathbf{G}^{(d)}(j_d). \tag{5.59}$$

At the one hand, since the size of the full tensor $\mathbf{A}$ is $(n^{(1)}, n^{(2)}, \ldots, n^{(d)})$, the size of the mode-1 matricization is $(n^{(1)}, \prod_{k=2}^{d} n^{(k)})$. Thus, the matrix U is small and its QR-decomposition may be direct and cheap. At the other hand, the matrix V is very large and its QR-decomposition requires some structured procedures. In the case the decompositions

$$U = Q_u R_u, \quad \text{and} \quad V = Q_v R_v, \tag{5.60}$$

are available, we can build a small matrix $P = R_u R_v^T$ of size $r \times r$ whose SVD is easy to compute:

$$P = X\Sigma Y^T. \tag{5.61}$$

The matrix Σ is diagonal with size $\hat{r} \times \hat{r}$ where $\hat{r} \leq r$ is obtained after truncation [4] while X and Y are of size $r \times \hat{r}$ with orthonormal columns. For the matrix $A^{(1)}$, matrices of dominant singular vectors are $\hat{U} = Q_u X$ and $\hat{V} = Q_v Y$.

Since the QR-decomposition of U might be direct, let us now describe how the QR-decomposition of V can be done structurally. For this purpose, its needed to orthogonolize some factors. The lemma below from [80, p.2302] denotes that for TT-decomposition with orthogonal cores, the corresponding matrices have orthonormal rows.

Lemma 5.16

If a tensor $\mathbf{Z}$ *is expressed as*

$$\mathbf{Z}(\alpha_1, j_2, \ldots, j_d) = Q^{(2)}(j_2) Q^{(3)}(j_3) \ldots Q^{(d)}(j_d), \tag{5.62}$$

where $Q^{(k)}(j_k)$ *is a* $r_{k-1} \times r_k$ *matrix,* $k = 1, 2, \ldots, d$, *and* $r_d = 1$ *(for fixed* $j_k, k = 2, ..., d$, *the product reduces to a vector of length* r_1, *which is indexed by* α_1*), and the matrices* $Q^{(k)}(j_k)$ *satisfy orthogonality condition*

$$\sum_{j_k} Q^{(k)}(j_k) \left(Q^{(k)}(j_k)\right)^T = I_{r_{k-1}} \tag{5.63}$$

where $I_{r_{k-1}}$ *is the identity matrix of size* r_{k-1}, *and* $\mathbf{Z}$, *considered as a matrix* Z *of size* $r_1 \times \prod_{k=2}^{d} n^{(k)}$, *has orthonormal rows:*

$$(ZZ^T)_{\alpha_1 \hat{\alpha}_1} = \sum_{j_2, \ldots, j_d} Z(\alpha_1, j_2, \ldots, j_d) Z(\hat{\alpha}_1, j_2, \ldots, j_d) = \delta_{\alpha_1 \hat{\alpha}_1}.$$

By using this lemma, one can address a structured QR-decomposition of the matrix V given in TT format. Consider

$$V(j_2, \ldots, j_d) = \mathbf{G}^{(2)}(j_2) \ldots \mathbf{G}^{(d)}(j_d).$$

By an algorithm that sweeps all the cores from right to left, one may orthogonalize the cores $\mathbf{G}^{(k)}(j_k)$, $k = \{d, d-1, \ldots, 2\}$, following the steps below.

First, since $r_d = 1$, $\mathbf{G}^{(d)}(j_d)$ is a $(r_{d-1}, n^{(d)})$ matrix whose QR-decomposition writes

$$\mathbf{G}^{(d)}(j_d) = R^{(d)} Q^{(d)}(j_d),$$

where the size of $R^{(d)}$ is (r_{d-1}, r_{d-1}) and $Q^{(d)}(j_d)$ of size $(r_{d-1}, n^{(d)})$ has orthogonal rows:

$$\sum_{j_d} Q^{(d)}(j_d) \left(Q^{(d)}(j_d)\right)^T = I_{r_{d-1}}.$$

Then, setting

$$\mathbf{G}'^{(d-1)}(j_{d-1}) = \mathbf{G}^{(d-1)}(j_{d-1}) R^{(d)}$$

4. The truncation may be done by fixing $\hat{r}$ or following a threshold $\hat{\epsilon}$. In addition, notice that in this case $\hat{r}$ may be the sought smaller rank r' or may be reduced once more according to the threshold.

leads to a core tensor of the same size using core-contracted product with size transformation

$$\left(r_{d-2}, n^{(d-1)}, r_{d-1}\right) \times (r_{d-1}, r_{d-1}) \longrightarrow \left(r_{d-2}, n^{(d-1)}, r_{d-1}\right).$$

This allows us to write

$$V(j_2, \ldots, j_d) = \mathbf{G}^{(2)}(j_2) \ldots \mathbf{G}^{'(d-1)}(j_d) Q^{(d)}(j_d),$$

with $Q^{(d)}(j_d)$ that satisfies (5.63).

By induction we may suppose at the step $s = k+1$, that the cores $(d, d-1, \ldots k+1)$ are othogonal. Thus we have the representation

$$V(j_2, \ldots, j_d) = \mathbf{G}^{(2)}(j_2) \ldots \mathbf{G}^{'(k)}(j_k) Q^{(k+1)}(j_{k+1}) \ldots Q^{(d)}(j_d), \tag{5.64}$$

where matrices $Q^{(s)}(j_s)$, $s = k+1, \ldots, d$, satisfy (5.63) and it can be shown that at step $s = k$, the transformation still hold. For this purpose, it suffices to compute

$$\mathbf{G}^{'(k)}(j_k) = R^{(k)} Q^{(k)}(j_k) \tag{5.65}$$

with

$$\sum_{j_k} Q^{(k)}(j_k) \left(Q^{(k)}(j_k)\right)^T = I_{r_{k-1}}, \tag{5.66}$$

and to observe that the matrix $R^{(k)}$ of size (r_{k-1}, r_{k-1}) is independent of j_k.

This leads to the conclusion that $Q^{(k)}$ and $R^{(k)}$ can be computed via the orthogonalization of rows of the matrix G obtained by reshaping $\mathbf{G}^{(k)}$ into a matrix of size $r_{k-1}, n^{(k)} r_k$. Thus, we have computed the QR-decomposition using the core tensors $\mathbf{G}^{(k)}(j_k)$ of the TT-decomposition of the matrix A. To perform the compression, one may compute the compressed SVD and contract two cores containing common index set (e.g, α_1 in (5.44)) with two small matrices.

The whole process is presented in Algorithm 5.1 from [80]. In this algorithm the notation QR_{rows} denotes a QR factorisation with a Q-factor that has orthonormal rows. By SVD_δ, we denote a singular value decomposition where all singular values smaller that δ are set to zero.

Algorithm 5.1 (*TT-rounding*)

Inputs: d-dimensional tensor $\mathbf{A}$ *with cores* $\mathbf{G}^{(k)}$ *and accuracy* ε
Output : $\mathbf{B}$ *in TT-format with TT-ranks* r_k *via new cores* $\mathbf{G}^{(k)}$
These ranks r_k *are* δ*-ranks from matricized* $A^{(k)}$ *from* $\mathbf{A}$ *in mode* k
such that

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \varepsilon \|\mathbf{A}\|_F, \text{ and } \delta = \frac{\varepsilon}{\sqrt{d-1}} \|\mathbf{A}\|_F.$$

Compute truncation parameter $\delta = \dfrac{\varepsilon}{\sqrt{d-1}} \|\mathbf{A}\|_F$

% **Orthonormalization**

for $k = d : -1 : 2$

$[\mathbf{G}^{(k)}(\beta_{k-1}, j_k\beta_k), R(\alpha_{k-1}, \beta_{k-1})] = QR_{rows}G^{(k)}(\alpha_{k-1}, j_k\beta_k)$;
$\mathbf{G}^{(k-1)} = \mathbf{G}^{(k)} \times_3 R$;
end for
% ***Compression***
for $k = 1 : 1 : d-1$
$[G^{(k)}(\beta_{k-1}j_k, \gamma_k), \Lambda, V(\beta_{k-1}, \gamma_k)] = SVD_\delta(G^{(k)}(\beta_{k-1}j_k; \beta_k)$;
$G^{(k+1)} = G^{(k+1)} \times_1 (V\Lambda)^T$;
end for

After the first mode has been compressed, a same algorithm may be used to compute the structured QR-decomposition of U (and V) as above for other modes. Thus, the dimensionality reduction is performed in the computation of matrices R_U, R_v (see (5.60), of the QR-decomposition and in the compi-utation of singular values, of the reduced rank and of matrices X and Y in (5.61). However, we may avoid these decomposition for every mode from scratch by using information from previous steps (see [80] for more details). The number of operations required for TT-rounding is approximated to be $\mathcal{O}(dnr^3)$ if we assume $n^{(k)} \sim n$ and $r_k \sim r$ and estimating that each right-left QR-decomposition of the $nr \times r$ matrices costs $\mathcal{O}(nr^3)$ operations if we begin by a Tucker decomposition and apply the TT-decomposition to its core only, the complexity may be reduced to

$$\mathcal{O}(dnr^2 + dr^4)$$

where the first term is the cost of d-sequentiel QR-decomposition of Tucker factors.

We are ready to adress the solution of the linear system (4.9) for which the operator is a Laplacian and thus takes the form (5.53) and the right-hand side is given by (1.59) but is tensorized in the form (5.38).

5.3.3 Linear systems in TT format

We are concerned by a Conjugate Gradient solver but instead of storing matrices and vectors in their initial structures, the solution is sought in the TT-format. Consider a linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.67}$$

where both the matrix $\mathbf{A}$ and the right-hand side $\mathbf{b}$ are represented in Tensor-Train format. In this context, vectors of length $N = n^{(1)}n^{(2)} \dots n^{(d)}$ are considered tensors of size $n^{(1)} \times n^{(2)} \times \dots \times n^{(d)}$ with low TT-ranks and square matrices acting on them are tensors of size $n^{(1)}.n^{(1)} \times n^{(2)}.n^{(2)} \dots \times n^{(d)}.n^{(d)}$. By hypothesis, the system is very large. Thus, the exact solution might be very hard to obtain. The aim is to approximate the solution with a given accuracy using Krylov-like methods (here the CG) 2.2. The system above has to be treated as an overdetermined linear system by fixing all the cores but one:

$$\sum_{j_k=1}^{n^{(k)}} \mathbf{A}(i_k, j_k)\mathbf{x}(j_k) = \mathbf{b}(i_k), \;\; 1 \le i_k, j_k \le n^{(k)}, \tag{5.68}$$

where $\mathbf{A}(i_k, j_k)$ is a matrix of size $N^{(k)} \times N^{(k)}$ with $N^{(k)} = \prod_{s,s\neq k} n^{(s)}$ for each fixed (i_k, j_k) and vectors $\mathbf{x}(j_k), \mathbf{b}(i_k)$ are vectors of length $N^{(k)}$. Since we need all cores to

be fixed except one, $G^{(k)}$, there exists $\mathbf{w}(j_k)$ such that

$$\mathbf{x}(j_k) = Q\mathbf{w}(j_k), \tag{5.69}$$

where Q is a matrix of size $N^{(k)} \times r^{(k-1)}r^{(k)}$. The matrix Q might be made orthogonal or can be treated as a tensor of the form

$$\mathbf{Q}(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_d, \alpha_{k-1}\alpha_k)$$
$$= G^{(1)}(i_1) \ldots G^{(k-1)}(i_k, \alpha_{k-1}) G^{(k+1)}(\alpha_{k+1}, i_{k+1}) \ldots G^{d}(i_d).$$

To have the matrix Q with orthonormal columns, the cores $G_{s,\ s=1:k-1}$ have to be orthonormalized from the left while the cores $G_{s,\ s=k+1:d}$ are orthonormalized from the right.

The restriction (5.69) leads to smaller linear systems of the form

$$\sum_{j_k} (Q^T A(i_k, j_k) Q)\mathbf{w}(j_k) = Q^T \mathbf{b}(i_k), \tag{5.70}$$

called *local problem*, where each matrix

$$B(i_k, j_k) = Q^T A(i_k, j_k) Q \tag{5.71}$$

is smaller and has size $r_{k-1}r_k \times r_{k-1}r_k$ while the full matrix

$$B = \sum_{j_k} (Q^T A(i_k, j_k) Q) \tag{5.72}$$

in (5.70) has size $r_{k-1}r_k n^{(k)} \times n^{(k)} r_{k-1} r_k$. Each linear system in the sum (5.70) is called local problem. Matrices $B(i_k, j_k)$ can be computed from the TT-representation of $\mathbf{A}$ and $\mathbf{x}$ and a linear system has to be solved at each such step. Although B is small compared to the initial matrix, its size can still be large. Thus, it is advisable not to form explicitly the matrix B but to solve either this local problem by iterative solver such as the CG algorithm. The CG solver minimizes the quadratic function

$$\langle \mathbf{Ax}, \mathbf{x} \rangle - 2\langle \mathbf{b}, \mathbf{x} \rangle, \tag{5.73}$$

where $\langle ., . \rangle$ denotes the scalar product [5] in tensor format and $\mathbf{x}$ is associated with the tensor $\mathbf{x}(i_1, \ldots, i_d) = G^{(1)}(i_1)G^{(2)}(i_2) \ldots G^{(d)}(i_d)$ with small TT-ranks. Most of used algorithms for such systems use the Alternating Least Squares (ALS) method. For this method, all the cores are fixed except one. The minimization problem for each core is linear and may be solved iteratively with respect to one unknown and one alternates for each mode.

However, the ALS method requires all the TT-ranks to be known in advance and its convergence is known to be slow [94]. To tackle these limitations, Modified ALS (MALS) have been proposed [95, 96]. The MALS comes from the so called Density Matrix renormalization Groups (DMRG) [97]. The DMRG was proposed to look for minimal eigenvalues and wavefunctions of a quantum spin system with k spins.

Following the paper [94], the DMRG we are exploring here is an ALS-like algorithm where, instead of solving with respect to one core as in (5.68), one minimizes

5. This scalar product is reduced to Euclidean scalar product when matricizations are used.

over a pair of cores $G^{(k)}(i_k)G^{(k+1)}(i_{k+1})$ represented by a supercore using contraction over one common rank dimension (α_k omitted in the formulation below):

$$W(i_k, i_{k+1}) = G^{(k)}(i_k)G^{(k+1)}(i_{k+1}). \tag{5.74}$$

This is not formulated explicitly but is taken into account when formulating the local problem (5.70). This gives rise to contraction, compression and thus truncation while the algorithm explores less directions (modes). However, if $\mathbf{A}$ comes from a discretization of a high dimensional equation, the local problem may be larger since the one dimensional size n is not usually small. In such case, the Quantized TT-format is more appropriate. The QTT-format allows any vector of a univariate function on a grid with $2^l (l \in \mathbb{N}, l \geq 2)$ points to be transformed in a structured high dimensional array considering $n = 2^l$. Then, a vector can be considered as a d-dimensional QTT-tensor with mode sizes 2 by binary coding. Such tensorization can be generalized to any dimension arbitrarily.

Let us consider that a l variables function is discretized on a tensor grid with $n = 2^l$ points in each direction. After QTT-tensorization, let $r = \max_{k\in\{1,\dots,l\}}\{r_k\}$ be the maximal rank of TT-ranks. One can observe that, using iterative methods, local problems will have at most $4r^2$ unknowns depending on the ranks and not on the dimension l. Unfortunatly, this dependence on the TT-ranks grows exponentially, leading to the curse of ranks. Thus, truncation is required.

In addition to ranks growth in local calculations, local problems are often ill-conditioned and have to be preconditioned. Fortunately, we may be able to compute a preconditioner with sufficiently small complexity [94], since a matrix inversion with sufficient accuracy is enabled in TT-format. To make all the processus work in an algorithm, four key aspects have been pointed out in the reference paper [94].

Solve large systems iteratively, small ones exactly

The linear system (5.67) is transformed into the sum (5.70) where small equations are solved. In general, from (5.69), the matrix $[Qw(j_k)]_{j_k=1}^{n^k}$ is the full vector $\mathbf{x}$, of size $[n^{(1)}, n^{(2)}, \dots, n^{(d)}]$, reshaped into an $N^{(k)} \times n^{(k)}$ matrix, with $N^{(k)} = \prod_{l=1\ k\neq l}^{d} n^{(k)}$. Considering Q as an orthogonal projector and assuming the local residual

$$Q^T A(i_k, j_k)Qw(j_k) - Q^T b(i_k)$$

for a given j_k has norm reduced to a given threshold ϵ, one can verify that this local residual norm can not be greater than the global residual norm. In fact, the equation (5.67) can be rewritten

$$Q^T \mathbf{A}\mathbf{x} = Q^T \mathbf{b}$$

such that

$$\|Q^T \mathbf{A}\mathbf{x} - Q^T \mathbf{b}\| = \epsilon$$

where one can verify that

$$\|Q\|\|\mathbf{A}\mathbf{x} - \mathbf{b}\| \geq \epsilon.$$

Since $\|Q\| = 1$, one gets $\|\mathbf{A}\mathbf{x} - \mathbf{b}\| \geq \epsilon$ that shows that the global residual norm can not be less than the local one. For this reason, it is advised to solve local systems accurately. In addition, if TT-ranks are small enough (this is the case for Laplace-like operators), it is recommended to assemble the full matrix B in (5.72) of size

$n^2r^2 \times n^2r^2$ and to solve local systems by direct solvers. Then, when TT-ranks become larger ($n^2r^2 \geq 1000$ as proposed in [94]), iterative solvers have to be run. Fast convergence of this last is expected since it starts with a residual that is already small.

Truncation based on the residuals

In the DMRG algorithm, the supercore $W(i_k, i_{k+1}) = G^{(k)}G^{(k+1)}$ (see (5.74)) is optimized by construction via its SVD and the k^{th} TT-ranks can be determined adaptively from the SVD. However, the approximation is accurate only in the Euclidean norm (or its equivalent). For $\mathbf{A}$ with a large condition number (for example multidimensional Laplace operators or others from finite differences methods) and for the available approximation $\hat{\mathbf{x}}$ of $\mathbf{x}$ in the Frobenius norm, we have $\|\mathbf{x} - \hat{\mathbf{x}}\| \leq \epsilon$ that leads to a residual norm $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|$ that can be large. Thus, instead of approximating

$$W(i_k, i_{k+1}) = \hat{W}(i_k, i_{k+1}) = G^k(i_k)G^{k+1}(i_{k+1})$$

via adaptative SVD, one approximates with a fixed rank r that has to provide a local residual norm $\|B\hat{W} - \mathbf{b}\|$ not worse than the original. This means, the truncation is made by setting to zeros all singular values from $r+1$. The value of r may be chosen by trial and error such that the local residual norm is small. Of course, this requires additional matrix-vector products, but their cost is expected to be compensated by much improved convergence.

Random restart

Suppose all local residuals are reported to be small but the obtained solution is not adequate for the global system. This arises when the algorithm is trapped in local minima. Such a situation can be checked by computing the global residual norm $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|$. However, a cheaper way of checking is to compare $\langle \mathbf{A}\mathbf{x}, \mathbf{z}\rangle$ and $\langle \mathbf{b}, \mathbf{z}\rangle$ where $\mathbf{z}$ is a random vector with fixed TT-ranks (say,2). If the difference is large, the convergence is not obtained and the process has to restart.

Fast matrix-vector product computation

In the CG solvers, the matrix-vector product is the most expensive operation. Here, it means the product between a tensorized matrix by a tensorized vector in TT-format. Since the matrix $\mathbf{A}$ in TT-format can be expressed by its cores

$$\mathbf{A}(i_1, \ldots, i_d, j_1, \ldots, j_d) = \mathbf{G}^{(1)}(i_1, j_1) \ldots \mathbf{G}^{(d)}(i_d, j_d)$$

where $\mathbf{G}^{(k)}(i_k, j_k)$ are like parameter-dependent matrices of order $r_{k-1} \times r_k$, and similarly is the vector $\mathbf{x} = \mathbf{x}^{(1)}(j_1) \ldots \mathbf{x}^{(d)}(j_d)$, the matrix-vector product is the computation of the sum

$$\begin{array}{rcl} \mathbf{y}(i_1, \ldots, i_d) & = & \sum_{(j_1,\ldots,j_d)} \mathbf{G}^{(1)}(i_1, j_1) \ldots \mathbf{G}^{(d)}(i_d, j_d)\mathbf{x}^{(1)}(j_1) \ldots \mathbf{x}^{(d)}(j_d) \\ & = & \sum_{(j_1,\ldots,j_d)} (\mathbf{G}^{(1)}(i_1, j_1) \otimes \mathbf{x}^{(1)}(j_1)) \ldots (\mathbf{G}^{(d)}(i_d, j_d) \otimes \mathbf{x}^{(d)}(j_d)) \\ & = & \mathbf{y}^{(1)}(i_1) \ldots \mathbf{y}^{(d)}(i_d) \end{array}$$

where

$$\mathbf{y}^{(k)}(i_k) = \sum_{j_k} (\mathbf{G}^{(k)}(i_k, j_k) \otimes \mathbf{x}^{(k)}(j_k)). \tag{5.75}$$

The TT-ranks of $\mathbf{y}$ are the products of ranks for the TT-matrix and for the TT-vector. When the ranks are kept small, the computation of $\mathbf{y}^{(k)}$ can be realized by a matrix-by-matrix product. The summation over j_k is equivalent to the product of a matrix of size $r^2n \times n$ from $G^{(k)}(i_k, j_k)$ (obtained by reordering and reshaping the dimension) by a matrix of size $n \times r^2$ from $X^{(k)}(j_k)$. The complexity of this matrix-by-matrix product is $\mathcal{O}(n^2r^4)$ leading to $\mathcal{O}(dn^2r^4)$ for the total matrix-by-vector product. This remains reasonable as long as $r << n$.

Since after almost each operation the TT-ranks grow, truncation has to be applied. Application of the TT-rounding algorithm necessitates $\mathcal{O}(dnr^6)$ operations. For n very large, one may first apply the Tucker format to the matrix and the vector and then TT-decomposition is applied to the compressed core. Several techniques have been proposed to compute quickly the matrix-by-vector product [94], [98] and aim to avoid explosion of TT-ranks. These techniques often combine multiplication and rounding in one step [80]. If the approximate TT-ranks of a product are also $\mathcal{O}(r)$, the total cost of this matrix-vector algorithm is $\mathcal{O}(n^2r^4)$ and may be $\mathcal{O}(n^2r^2 + dr^6)$ if the Tucker format is used.

Algorithm 5.2 (*Matrix-vector product*)

Inputs: matrix $\mathbf{A}$ *with cores* $\mathbf{G}^{(k)}(i_k, j_k)$ *and vector* $\mathbf{x}$ *with cores* $\mathbf{x}^{(k)}(j_k)$.
Output : $\mathbf{y} = \mathbf{Ax}$;
for k = 1:d

$$\mathbf{y}^{(k)}(i_k) = \sum_{j_k} \mathbf{G}^{(k)}(i_k, j_k) \otimes \mathbf{x}^{(k)}(j_k);$$

end(For)

Fast dot product

The following algorithm allows efficient computation of the dot product.

Algorithm 5.3 (*Dot product*)

Inputs: Tensor $\mathbf{A}$ *with cores* $\mathbf{G}^{(k)}(j_k)$ *and tensor* $\mathbf{B}$ *with cores* $\mathbf{H}^{(k)}(j_k)$
Output : $w = \langle \mathbf{A}, \mathbf{B} \rangle$;
$v = \sum_{j_1} \mathbf{G}^{(1)}(j_1) \otimes \mathbf{H}^{(1)}(j_1)$;
for k = 2:d
$p_k(j_k) = v \left(\mathbf{G}^{(k)}(j_k) \otimes \mathbf{H}^{(k)}(j_k) \right)$;
$v = \sum_{j_k} p_k(j_k)$;
end(For) $w = v$.

After defining these operations, one gets a Conjugate Gradient algorithm in TT-tensor format as presented below with 2 new operations: contracted product and truncation in TT.

Algorithm 5.4 (*CG in TT-format*)

Given $x_0, \epsilon > 0$, *and max-rank* $r \geq 1$
$\mathbf{r}_0 \leftarrow \mathbf{A}x_0 - \mathbf{b}$;
$\mathbf{p}_0 \leftarrow -\mathbf{r}_0,\ k \leftarrow 0$;
While $\|r_k\| > \epsilon$
$\alpha_k \leftarrow \dfrac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{r}_k, \mathbf{A}\mathbf{r}_k \rangle}$; %*Compute the steplength*
$\mathbf{x}_{k+1} \leftarrow \mathcal{T}_r(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$; %*Update the truncated solution*
$\mathbf{r}_{k+1} \leftarrow \mathcal{T}_r(\mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k)$ %*Update the truncated residual*
$\beta_{k+1} \leftarrow \dfrac{\langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle}{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}$ %$Ensure A-conjugate directions$
$\mathbf{p}_{k+1} \leftarrow \mathcal{T}_r(-\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k)$ %$Update the truncated descent direction$
$k \leftarrow k + 1$
End (while)

5.3.4 A solver for systems from 3D MIRP

Now let us solve a general linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.76}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$ and will be expressed in tensor format (for this reason, they are written with bold case).

The operator

The matrix (4.9) is a d-dimensional differential operator that can be written of the form (see Annexe (B.3)

$$\mathbf{A} = \left(\sum_{k=1}^{d} I_{n^{(1)}} \otimes \dots I_{n^{(k-1)}} \otimes A^{(k)} \otimes I_{n^{(k+1)}} \dots \otimes I_{n^{(d)}} \right), \tag{5.77}$$

where

$$A^{(k)} = \frac{1}{h^{(k)^2}} \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n^{(k)} \times n^{(k)}}.$$

Instead of positivity, one may use negativity by working with $-A^{(k)}$. As for matrices, the convergence rate of a CG-like algorithm is connected to the condition number of the tensor $\mathbf{A}$. Let the matrices $A^{(k)}$ be positive definite with eigenvalues $\lambda_1^{(k)} \geq \lambda_2^{(k)} \geq \dots \geq \lambda_{n^{(k)}}^{(k)} > 0$. Then, the condition number of the tensor $\mathbf{A}$ is given by

$$k(\mathbf{A}) = \sum_{k=1}^{d} k(A^{(k)}) = \frac{\sum_{k=1}^{d} \lambda_1^{(k)}}{\sum_{k=1}^{d} \lambda_{n^{(k)}}^{(k)}}. \tag{5.78}$$

In the simple case where $n^{(k)} = n$ for all k, the eigenvalues $\lambda_\mu^{(k)} = \lambda_\mu, 1 \le \mu \le n^{(k)}$ for each matrix $A^{(k)}$ and the following remark from [6, p.467] presents the relation between the condition number of $\mathbf{A}$ and of the matrix $A^{(k)}$.

Remark 5.1

The condition number of the tensor $\mathbf{A}$ *depends on the numbers* $n^{(k)}$ *but not on the dimension* d*. In particular, the following inequalities hold:*

$$\min_k k(A^{(k)}) \le k(\mathbf{A}) \le \max_k k(A^{(k)}). \tag{5.79}$$

This remark encourages enlargement of the dimension d (when it is possible) while decreasing the matrix-sizes $n^{(k)}$. And from this point of view, the QTT-solvers are recommanded.

One can observe that this operator is general and not related to the application at hand, the image registration here. However, the size of the operator is derived from the size of the images to be registered. All the important information on images for registration is carried in the right-hand side $\mathbf{b}$. Thus, a particular attention has to be paid to this and this is developed below.

The right-hand side

In general, the image is stored in a long vector of length $N = \prod_{k=1}^{d} n^{(k)}$ and indices $1 \le j \le N$ are used to localize each image intensity and each grid point. Using the isomorphism $j \Leftrightarrow (j_1, \ldots, j_d)$, let us consider the vectorized grid points $x_j = a + jh, \quad h = \dfrac{b-a}{N+1}$, where $[a, b]$ is the domain of the vectorized image. The intensities are values $u(j) = f(x_j)_{\,j=1:N}$ on a uniform grid. Assuming each $n^{(k)}$ is a power of 2, there exists $p \in \mathbb{N}$ such that $N = 2^p$. Thus the indices $j = j_1 + 2j_2 + \ldots + 2^{p-1}j_p$ can be reshaped into a QTT-tensor by the formula [99]

$$u(j) = f(x_j) = f(a + jh) = f(a + (j_1 + 2j_2 + \ldots + 2^{p-1}j_p)h). \tag{5.80}$$

For $t_k = \dfrac{a}{p} + 2^k j_k h$ (using the trick $a = p\dfrac{a}{p}$) we have

$$f(x_j) = f(t_1 + t_2 + \ldots + t_p).$$

For example, observe that $t_1 + t_2 + \ldots + t_p$ for $p = 4$ gives

$$\begin{aligned} t_1 + t_2 + t_3 + t_4 &= \begin{pmatrix} t_1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t_2 + t_3 + t_4 \end{pmatrix} \\ &= \begin{pmatrix} t_1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ t_2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t_3 + t_4 \end{pmatrix} \\ &= \begin{pmatrix} t_1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ t_2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ t_3 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t_4 \end{pmatrix} \end{aligned}$$

that is in QTT-format. These grid points can then be approximated by functions such as linear functions $f(x) = \lambda x = \lambda(t_1 + t_2 + \ldots + t_p)$, exponential functions $f(x) = \exp(\lambda x) \exp^{(\lambda t_1)} \exp^{(\lambda t_2)} \ldots \exp^{(\lambda t_p)}$, polynomial functions $f(x) = x^q = (t_1 + t_2 + \ldots + t_p)^q$ or a combination of them.

For non-parametric image registration, the function to be evaluated to these grid points (see 1.59) is

$$b(x_j) = \beta[(I_f\Phi^{-1} - I_m)\nabla I_m](x_j)$$

where β is a constant that may take into account some statistical parameters in the images (for example the mean, the dispersion or some normalization values). I_f and I_m are respectively the fixed and moving images interpolated via spline functions while $\Phi = Id + u$ is the sought transformation. Remember $\Phi^{-1} = Id - u$ where u is the computed displacement field. The notation dI_m denotes the derivatives of the moving image. Thus, by using TT-tensor format, an internal truncation is made and may be considered as regularization of the images. However, some images may be more sensitive to such truncation, as will be observed by numerical experiments in Chap 6.

5.3.5 Preconditioning

For most structured matrices, the inverse approximation is expected to be cheap. Thus, the privileged preconditioner is the approximate inverse, provided that this will preserve the structure and the sparsity patterns of the matrix. Here we present three ways of computing the inverse (two approximations and an exact computation for QTT Laplacian), but our algorithm use especially the approximation by exponential sum.

Preconditioning via Exponential Sums Inverse (ESI)

Consider a matrix A such that its spectrum, denoted $\sigma(A)$, is contained in the left complex halfplane:

$$\sigma(A) \subset \{z \in \mathbb{C} : Re(z) < 0\}.$$

It is easy to verify that

$$A\left(-\int_0^\infty \exp(tA)dt\right) = -\int_0^\infty A\exp(tA)dt = -\int_0^\infty \frac{d}{dt}\exp(tA)dt = \exp(0A) = I.$$

Thus,

$$A^{-1} = -\int_0^\infty \exp(tA)dt. \tag{5.81}$$

Let us now consider the Kronecker tensor (this can be considered as a large matrix) $\mathbf{A}^{-1}$, the inverse of $\mathbf{A}$ that can be approximated by extending (5.81) (see [98]) to multidimensional case. This is given by

$$\mathbf{A}^{-1} = -\int_0^\infty \bigotimes_{k=1}^{d} \exp(tA^{(k)})dt, \tag{5.82}$$

since

$$\exp(t\mathbf{A}) = \exp\left(t\sum_{k=1}^{d} A^{(k)}\right) = \prod_{k=1}^{d} \exp\left(tA^{(k)}\right) = \bigotimes_{k=1}^{d} \exp\left(tA^{(k)}\right).$$

The preconditioner $\mathbf{M}$ we are computing is an approximation of $\mathbf{A}^{-1}$ performed by quadrature (formula of Stenger [98]) of the integral (5.82). This is given in the following theorem

Theorem 5.17

Consider a tensor $\mathbf{A}$ *given by the formula* (5.77) *with the spectrum contained in* $\mathbf{\Omega} = [\lambda_{min}, \lambda_{max}] \oplus i[-\mu, \mu] \subseteq \mathbb{C}_- = \{z \in \mathbb{C} : Re(z) < 0\}$. *Let* Γ *denote the boundary of* $-[1, \Lambda + 1] \oplus i[-\mu - 1, \mu + 1]$ *For* $k, j \in \mathbb{N}$ *and*

$$h_{st} = \frac{\pi^2}{\sqrt{k}}, \tag{5.83}$$

$$t_j = \log\left(\exp(jh_{st}) + \sqrt{1 + \exp(2jh_{st})}\right), \tag{5.84}$$

$$w_j = \frac{h_{st}}{\sqrt{1 + \exp(-2jh_{st})}}, \tag{5.85}$$

the tensor

$$\mathbf{M} = -\sum_{j=-k}^{k} \frac{2w_j}{\lambda_{min}} \bigotimes_{i=1}^{d} \exp\left(\frac{2t_j}{\lambda_{min}} A^{(i)}\right) \tag{5.86}$$

fulfils

$$\|\mathbf{A}^{-1} - \mathbf{M}\| \leq \frac{C_{st}\|\mathbf{A}\|}{\pi\lambda_{min}} \exp\left(\frac{2\mu\lambda_{min}^{-1} + 1}{\pi} - \pi\sqrt{2k}\right) \times \oint_{\Gamma} \left\| \left(\lambda I - \frac{2}{\lambda_{min}}\mathbf{A}\right)^{-1} \right\| d_{\Gamma}\lambda. \tag{5.87}$$

An other interesting algorithm for approximating the inverse is the Newton-Shulz algorithm.

Preconditioner via Newton-Schulz algorithm

The Newton-Schulz algorithm is an iterative approximation of the inverse $\mathbf{M}$ (for matrices [76, p.75] and for tensors [6, 100, 101, p.412]) via a fixed point iterations described below. The algorithm stops when the norm $\|\mathbf{I} - \mathbf{A}\mathbf{M}_k\|$ is becomes less than a given threshold. A quadratic convergence to $\mathbf{A}^{-1}$ is reported whenever $\|\mathbf{I} - \mathbf{A}\mathbf{M}_k\| < 1$

Algorithm 5.5 (*Newton-Schulz*)

$0 < \alpha \in \mathbb{R}$;
$\mathbf{M}_0 = \alpha\mathbf{I}$;
For $k = 1, ...$
$\quad \mathbf{M}_k = T_{r,\epsilon}(\mathbf{M}_{k-1}(2\mathbf{I} - \mathbf{A}\mathbf{M}_{k-1}))$
end(For)

A modified approach has been proposed by Oseledets & all [101] where the truncation is adapted and gives the following algorithm.

Algorithm 5.6 (*Modified Newton-Schulz*)

Choose an initial guess $\mathbf{M}_0 \approx \mathbf{A}^{-1}$
$\mathbf{Y}_0 = \mathbf{A}\mathbf{M}_0$
For
$\mathbf{H}_k = T_{r,\epsilon}(2\mathbf{I} - \mathbf{Y}_k)$
$\mathbf{Y}_{k+1} = \mathbf{T}_{r,\epsilon}(\mathbf{Y}_k\mathbf{H}_k)$
$\mathbf{M}_{k+1} = T_{r,\epsilon}(\mathbf{M}_k\mathbf{Y}_k)$
end(For)

Theoretically, in an ideal case, we will have $\mathbf{H}_k \to \mathbf{I}, \mathbf{Y}_k \to \mathbf{I}$, and $\mathbf{M}_{k+1} \to \mathbf{A}^{-1}$. However, in Newton-Shulz algorithms a matrix-by-matrix product is required at each step. If this can be feasible for some structured matrices, these algorithms are recommended. In most cases, it becomes infeasible, in particular when the TT-ranks of the product become larger and larger. Explicit inverse has also been proposed for multidimensional Laplace operators.

Preconditioner via Explicit Inversion(EI)

Consider notations and matrices defined in (5.47), then it can be shown that (see [81]) for $(n = [2^l, 2^l]\ l \geq 3)$ we have:

$$(\Delta^l)^{-1} =$$

$$\begin{bmatrix} E & L & E^T & E \end{bmatrix} \bowtie \begin{bmatrix} I & L & E^T & E \\ & 2N & & \\ & L+E^T & N & \\ & L+E & & N \end{bmatrix}^{\bowtie(k-2)} \bowtie \begin{bmatrix} N+L \\ 2N \\ N+L+E^T \\ N+L+E \end{bmatrix}.$$

The Preconditioned (PCG) in Tensor-Train format is presented below:

Algorithm 5.7 (*PCG in TT-tensor representation*)

Given $\mathbf{x}_0$, $\epsilon > 0$ $\mathcal{M}$ % *A preconditioner*

$\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$; $\mathbf{z}_0 = \mathcal{M}\mathbf{r}_0$; $\mathbf{p}_0 = -\mathbf{z}_0$, $k = 0$;
While $\|r_k\| > \epsilon$
$\alpha_k = \dfrac{\langle \mathbf{r}_k, \mathbf{p}_k \rangle}{\langle \mathbf{p}_k, \mathcal{A}\mathbf{p}_k \rangle}$
$\mathbf{x}_{k+1} = \mathcal{T}_r(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$
$\mathbf{r}_{k+1} = \mathcal{T}_r(\mathbf{b} - \mathcal{A}\mathbf{x}_{k+1})$
$\mathbf{z}_{k+1} = \mathcal{M}\mathbf{r}_{k+1}$,
$\beta_{k+1} = \dfrac{-\langle \mathbf{z}_{k+1}, \mathcal{A}\mathbf{p}_k \rangle}{\langle \mathbf{p}_k, \mathcal{A}\mathbf{p}_k \rangle}$
$\mathbf{p}_{k+1} = \mathcal{T}_r(\mathbf{z}_{k+1} + \beta_{k+1}\mathbf{p}_k)$
$k \leftarrow k + 1$
End (while)

5.4 FA4DMIR algorithm

In this section we present the algorithm proposed in this thesis. We propose to call the algorithm *Flexible Algorithm For Deformable Medical Image Registration*

(FA4DMIR). This algorithm is based on *Flexible Algorithms for Image Registration (FAIR)* package from Jan Modersitzki [10]. Although our study focused on $3D$ images, $2D$ images can be registered by this algorithm. One is allowed to run the registration in a multilevel approach from a given minimal level l_{min} parameter (1.4.1).

The main features we have introduced in the general nonparametric FAIR-algorithms (see [10, p.137] include the way of solving linear systems and the way of computing the displacement field.

For linear systems, FA4DMIR algorithm enables two main extensions. First, it allows to use polynomial preconditioners 4.4, in particular the Tchebychev polynomial preconditioner that has proved to be effective for linear systems from image registration. These Neumann and Tchebychev polynomials preconditioners, are added to matrix splitting preconditioners that pre-exist in the FAIR package. Second, the FA4DMIR algorithm allows to the user to solve linear systems from image registration usig Tensor algorithms. In particular to solve linear systems within Tensor-Train format. This format allows efficient low-rank approximation 5.3 and efficient system solvers 5.3.3 that may lead to fast algorithms for large and high dimensional images. However, these images have to be less dense such that a sparse representation is allowed at a reasonable cost.

According to the displacement field large diffeomorphic deformations we have included the possibility a diffeomorphic transformation using the Large Deformation Difeomorphic Metric Mapping (LDDMM) (see Annexe E). The efficiency of linear system solvers is decisive for the efficiency of the registration algorithm. The FA4DMIR algorithm is flexible both because it inherits the FAIR package and it is flexible in the choice of parameters. A structure of parameters is provided to allow one the choice of wether the registration may be done in matrix or tensor format and the user can choose the preconditioner (SGS, Ichol, Jacobi,Tchebyshev or Neumann for matrices (see Chapter 4.2) and Newton-Schulz, ESI or EI (see Section 5.3.5) for tensors).

The user supplies the tolerance on the variable $tolX$, on the function $tolJ$, on the gradient $tolG$ and an expected final value for the functional $Jstop$.

A more diffeormorphic displacement can be obtained by successive compositions. However, this is facultative in FA4DMIR since, the more one wants a diffeomorphic displacement, the less there is a reduction in the function value and this induces a low convergence rate. To compute a diffeomorphic displacement, one uses an averaging interpolation (command interpn in Matlab) and this is implemented in FA4DMIR by the function `velocity2displacement`. The number of time steps T has to be supplied by the user. Then, setting $\phi^{\frac{1}{T}} = X + \frac{1}{T}\mathbf{v}$ where X is the initial grid and $\mathbf{v}$ comes from the linear solver, one loops on the time steps. The large deformation is thus considered as the composition of a series of T small deformations.

A pseudo-code of the algorithm is presented below while the organisation of the code is explained in Annexe G.

Algorithm 5.8 (*FA4DMIR*)

Input : I_f *and* I_m *% reference and moving images*
For $l_{min} : l_{max}$ *% loop on different levels*
Interpolate images within the grid (cubic spline);
Compute J, dJ *and* d^2J;
Evaluate stopping criteria
While *not (stopping criteria)*
% PCG in TT-Tensor (recommended) or keep matrix format
Solve $\mathbf{Av} = \mathbf{b}$*;*
% loop to get diffeomorphic displacement from velocity;
ϕ = *velocity2displacement*$(\mathbf{v}, T)$*;*
Compute α; *% by Line search strategy*
$\Phi_{k+1} = \Phi_k + \alpha\phi$*;*
Update I_m *and* ∇I_m;
Update J, dJ *and* d^2J;
end (While)
End (For)

The stopping criteria in FA4DMIR relies on the following behaviour:

1. Less change in the objective function value $(Jold - Jc) <= tolJ * (1 + |Jstop|)$
2. Less change in our variable $|\Phi_{old} - \Phi_{current}| <= tolX * (1 + norm(\Phi_{current}))$
3. Gradient in absolute value is less than a tolerance fixed on it? $|dJ| <= tolG * (1 + abs(Jstop))$
4. At a stationary point $norm(dJ) <= eps = 1e - 12$
5. Maximum number of iterations $iter >= maxIter$

The stopping criteria *stop* becomes true if $1, 2, 3$ are simultaneously verified or if one of 4 and 5 is verified.

Numerical experiments presented in the following Chapter 6 illustrates how better is FA4DMIR with respect to the other algorithms in the FAIR package.

Chapter 6

Numerical experiments in 3D MIRP

Since the focus of this thesis is on speeding up deformable 3D medical image registration process, this chapter aims to compare three registration algorithms with respect to their earlier behaviour and the computation time. This means that we are not interested in finding the minimizer of our functional (1.65), since this may be very hard to obtain, but in the reduction of the function value for a given budget in term of iterations and time. A particular focus is on the FA4DMIR algorithm, since this is one of the major contributions of this thesis. In what follows, the function value indicates the value of the functional (1.65). We present our three registration algorithms in Section 6.1 and we benchmark them therein via a perfomance profile based on the earlier decrease of the function value. Then, in Section 6.2, we present the dependency of FA4DMIR of the fixed maximal TT-rank and in Section 6.3, we present simulations for the three algorithms on some of our images presented in 3 where each level of each image is considered as a particular problem. A conclusion ends the chapter.

6.1 Performance profile of 3 deformable registration algorithms

The algorithms

The three algorithms we present here can be considered as three versions of a deformable nonparametric registration algorithm from the FAIR package. However, they are considered as three different algorithms since they differ either in the way of approximating the Hessian of the functional we are minimizing or in the way of solving the linear system induced by the process to get the search direction 2.1.2 in the optimization process.

The first algorithm, denoted *FAIR*, is the registration algorithm that uses the

approximation

$$A = D^T D \approx \nabla^2 F$$

where D is the differential operator described in annexe B. The linear system is solved by a PCG solver with Symmetric Gaus-Seidel (SGS) or Tchebychev polynomial preconditioner 4.4. This algorithm has the advantage that it uses a matrix-based operations that are well known. In addition, it is a differential operator, that remains constant during iterations. Thus, this matrix is computed and stored once and thus the problem is a multiple second-hand systems to be solved (see annexe (D.1)). The disadvantage of this algorithm is that the Hessian approximation may be inefficient since it may not catch second order information for the functional (1.65). This infers to it the slow convergence rate of first-order methods.

The second algorithm, denoted by *FA4DMIR*, approximates the Hessian as this former (FAIR algorithm)

$$A = D^T D \approx \nabla^2 F$$

but solves the linear system in TT-tensor format 5.3.3. The advantage with respect to the above algorithm is the possibility to compress and truncate certain operations such that the computation time is reduced for very large systems.

Finally, the third algorithm, denoted by *FAIR-GN*, approximates better the Hessian, following the Gauss-Newton procedure (see subsection 2.3):

$$A = J^T J + D^T D \approx \nabla^2 F.$$

where J is the Jacobian matrix from the residual functions (2.44). This algorithm is expected to be the most effective both in terms of rate of convergence and in terms of precision but also the most expensive in terms of time consumption since each iteration is the most expensive. This algorithm is recommended when the system is not very large and if the second-order information is sufficiently captured by a certain use of the Jacobian. However, it necessitates computation and storage of the Jacobian matrix and may be more sensitive to errors. Thus it may be slower and may suffer from error accumulation for very large systems as shown in simulations below.

In the remainder, we compare these three algorithms on our set of eleven couples of 3D medical images 3.1. As stated above, since the functional (1.65) to be minimized is expensive, we are not interested in a convergence test related to finding a local minimizer but simply in the early decrease of the function value by our algorithms.

Benchmarking

In this subsection we aim to evaluate and compare the three optimization algorithms (FAIR, FAIR-GN and FA4DMIR) using the performance profile process. The performance profile is considered here as a tool that enables a benchmarking process and allows analysis of the performance of each algorithm on the set of images given a limited computational budget in terms of time and memory. By plotting the best function value with respect to the number of functional evaluations, we compare the trajectories of these algorithms. This allows us to determine the algorithm that offers the most important reduction with the defined computational budget and the convergence test.

Here, the set of benchmark problems $\mathcal{P}$ is formed by different levels of our images 3.1 and thus

$$n_p = \#\mathcal{P} = 54.$$

Second, the set of optimization solvers (algorithms) $\mathcal{S}$ is defined here by the three algorithms described above:

$$n_s = \#\mathcal{S} = 3.$$

Third, we define below a convergence test.

Since the deformable registration problem on 3D images has expensive functional evaluations, we are interested in the convergence behaviour measured by the decrease of the function value within early iterations. This follows the convergence test (see [48, p.2]):

$$F_c \leq F_L + \tau\,(F_0 - F_L) \tag{6.1}$$

where $0 < \tau \leq 1$ is a fixed tolerance, F_0 is the starting function value, F_c is the current function value and F_L is considered as the smallest value of he functional F reached by any of the three algorithms within a given number μ_f of iterations. The convergence test (6.1) can also be written

$$(F_0 - F_c) \geq \eta\,(F_0 - F_L)\,. \tag{6.2}$$

In this last formulation, the current reduction is requested to be at least $(F_0 - F_c)$ $\eta = 1 - \tau$ times the best possible reduction $(F_0 - F_L)$. This is equivalent to

$$\left(1 - \frac{F_c}{F_0}\right) \geq \eta \left(1 - \frac{F_L}{F_0}\right). \tag{6.3}$$

For example, the convergence test (6.1) with $\tau = 10^{-1} \Rightarrow \eta = 0.9$ may represent a modest reduction of 90% of the total possible reduction. This is used as performance profile test since no algorithm may be able to reach an accurate estimate of F at a local minimizer.

The results describe a short-term behaviour of our three algorithms FAIR, FAIR-GN and FA4DMIR on the set $\mathcal{P}$. The maximal number of functional evaluations is $\mu_f = 100$, the maximal time is set to 8.410^4 (one day) and other limit resources are described in Table 3.2.

The function values generated by the solver s on problem p are in a column vector $h_{s,p} \in \mathbb{R}^{\mu_f}$. If, at iteration $k < \mu_f$, the solver s has verified the performance test, then all the successive function values $F(u_l), k < l \leq \mu_f$ are set to $F(u_k)$. In this case, $F(u_k)$ is the best function value produced by solver s after k iterations. These vectors are gathered into a 3-dimensional array A where the first slice (matrix) is formed by the history of the solver 1 to each problem (one column vector by problem), the second slice for the solver 2 and the third slice for solver 3. For each problem, F_L was taken as the minimal function value achieved by any solver after at most μ_f itarations.

Figure 6.1 presents the performance profile for our three optimization algorithms with the convergence test (6.3) where $\eta = 0.1$. Consider at one hand $\alpha = 1$ and observe that $\rho_{FAIR}(1) \approx 27\%$, $\rho_{FA4DMIR}(1) \approx 30\%$ and $\rho_{FAIR-GN}(1) = 52\%$. This means that the FAIR-GN algorithm has the best performance profile since it verifies the convergence test (where F_L is fixed to $0.51e-2$) earlier on 52% of the 54 problems. However, none of the algorithms verifies the convergence test on the whole set of problems. The reason is that, for very large images, the budget in terms of time and memory requirements was not suffiscient. The algorithms could not work on some image high levels. At the other hand, if we consider $\alpha = 8$ it can be seen that $\rho_{FA4DMIR}(8) \approx 60\%$, $\rho_{FAIR}(8) = 51\%$ and $\rho_{FAIR-GN}(8) = 55\%$. This means that, each of the algorithms $FAIR$ and $FAIR-GN$ requires 8 times the number

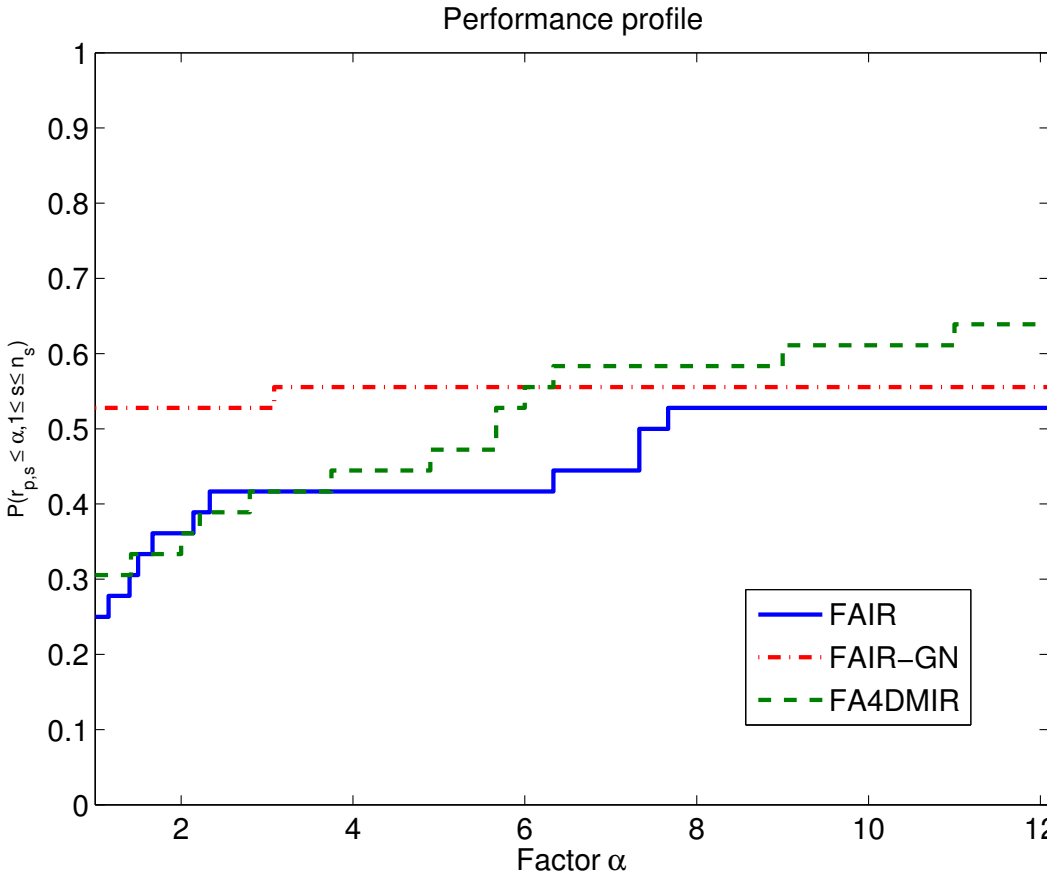


Figure 6.1: Comparison of the performance profiles between FAIR, FAIR-GN and FA4DMIR on a set of 54 3D images. The factor α is such that a given algorithm requires at least α times the efforts of the best algorithm to verify the convergence test.

of functional evaluations as $FA4DMIR$ on respectively (roughly) 9% and 5% of problems.

As proposed in [49], it makes sense to analyze the performance profiles of two by two algorithms separately.

Figure 6.2 presents the comparison of the performance profiles between The FAIR and the FA4DMIR algorithms with the convergence test (6.3). Here again $\eta = 0.1$ one can observe that FA4DMIR dominates FAIR in both convergence rate and robustness.

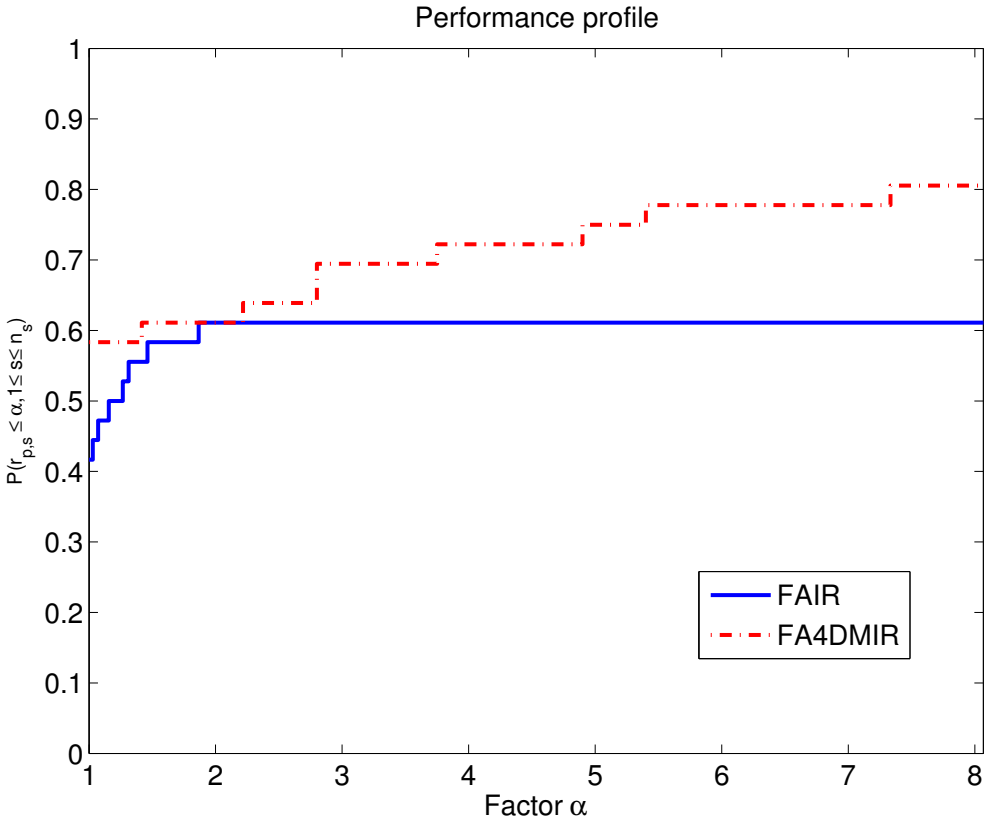


Figure 6.2: Comparison of the performance profiles of FAIR and FA4DMIR on a set of 54 3D images.

Figure 6.3 presents the comparison of the performance profiles between FAIR and FAIR-GN (top) and shows that FAIR-GN performs better than FAIR whereas FAIR is more robust since it reaches the tolerance value F_L on roughly 80% where FAIR-GN reaches it on 70% of problems. On the right, a comparison of the performance profiles between FAIR-Gn and FA4DMIR shows that FAIR-GN performs better that FA4DMIR, but this last is more robust since it reaches the tolerance value F_L on roughly 75% problems against 70% for FAIR-GN.

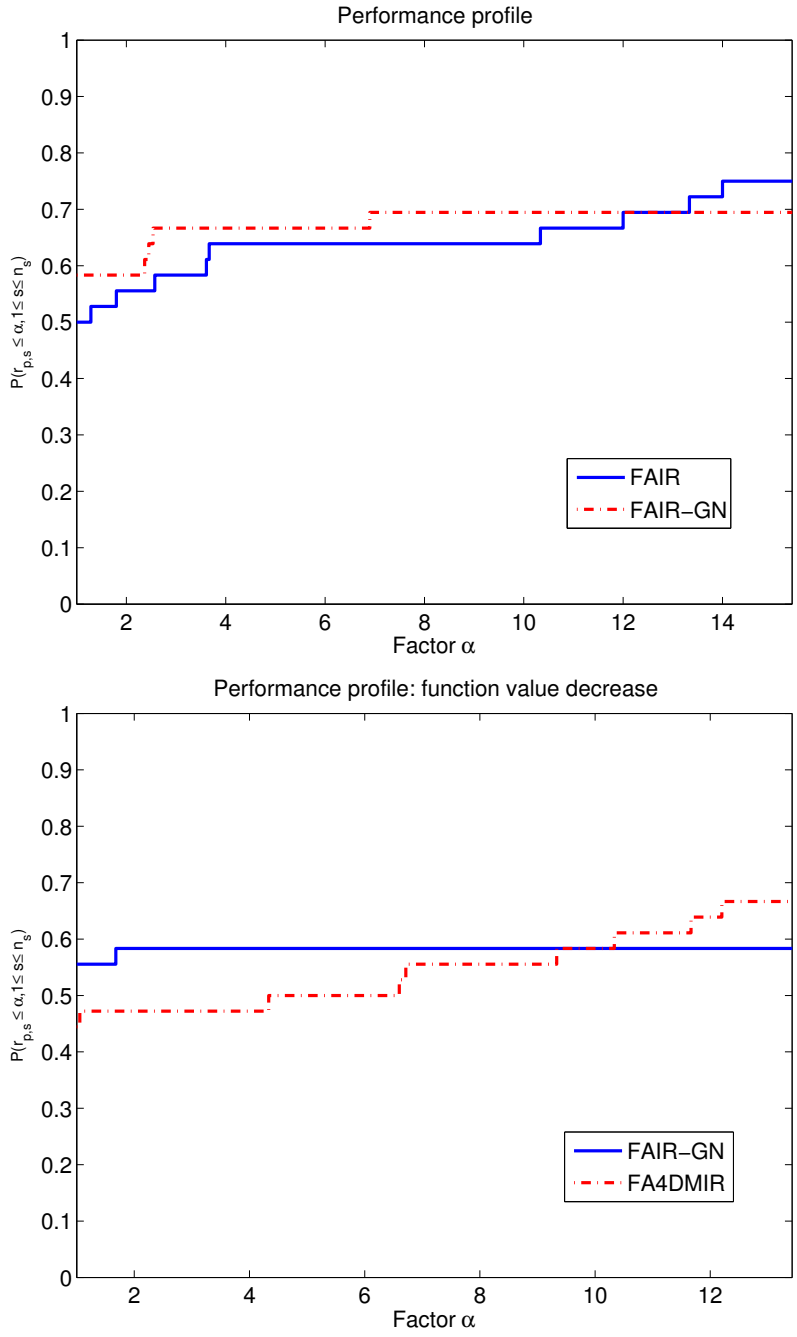


Figure 6.3: Comparison of performance profiles between FAIR and FAIR-GN (top) and between FAIR-GN and FA4DMIR (bottom) on a set of 54 3D images.

6.2 FA4DMIR and TT-rank

In this subsection we aim to highlight that one of the key element for a good use of tensor algorithms in TT formats is the truncature. When the truncature is based on a fixed destination rank (see 5.3.2), the choice of the destination rank (maximal rank for truncature) is not obvious. However, one may use certain indications from the literature.

6.2.1 FA4DMIR and TT-rank dependency

As stated in [6, p.475] there exists two strategies to perform tensor operations for optimisation problems with respect to truncation.

At the one hand, one may perform tensor operations within a truncation strategy constrained to minimize the approximation relative error. In this case, the tensor rank (here TT-rank) of the solution is determined adaptatively.

At the other hand, one may fix the tensor-rank and try to optimize the parameters with this fixed rank. It is this second strategy we use in this thesis. To solve the linear system (5.67), one needs an exponential decrease of the error to the solution. For this purpose, good approximations lead to good convergence rate. A bound on TT-ranks of Laplacian-like operators is known (see [98] or [6, 424]). Thus, when the mode size $n^{(k)}$ in each direction is relatively small, the system operator (left-hand) is well approximated. Then, the behaviour of the algorithm depends more on the structure of the right-hand side b.

According to L. Grasedyck in [98], the right-hand side given by

$$\mathbf{b} = \otimes_{k=1}^{l} b^{(k)}, \quad b^{(k)} \in \mathbb{R}^{n^{(k)}} \tag{6.4}$$

is required to be smooth and sparse to allow exponential decrease of the error.

For the first requirement, the smoothness, the right-hand side in our application is smooth since it is formed by the product of the residual functions of interpolated images and the derivatives of the moving image. The images are cubic spline functions that are at least twice differentiable. Thus, the right hand side can be considered as a grid function $\mathbf{b}(x_1, \ldots, x_l)$ that may be approximated by a product of low Kronecker-rank unidimensional functions (see 5.3.4)

$$\tilde{b}(x^{(1)}, \ldots, x^{(l)}) = \prod_{k=1}^{l} f_j^{(k)}(x^{(k)})$$

with an exponentially decaying error [98]. However, as noted by Grasedyck, when the dimension $l \to \infty$ the decay rate of the error tends to 1 and the approximation may become poorer. Thus, this has to be controlled.

For the second requirement, while it is almost certain to have sparse right-hand side, here the sparsity patterns depend on each image. Then, we are based on what is stated in [98]. That is, if the tensor vector $\mathbf{b}$ has m nonzeros entries with $m << N = \prod_{k=1}^{l}$ it can be decomposed by m tensor vectors that can be approximated by low rank tensors. Thus, less the right-hand side has nonzero entries, better its approximated by low-rank TT tensor.

An additional element, that is specific to the application in this thesis, is the sensitivity to truncation for some images. As we presented roughly in the previous chapter 5.3.4, MRI images are more sensitive to truncature based on neglecting

smaller singular values and their associated singular vectors. The reason is that, the acquisition techniques are based on magnetic fields orientations that localize water and fat material in the body excited by radio waves. Thus, sensitivity to truncature and sparsity patterns may explain why the use of Tensors methods may better improve for certain registration problems and less for others, even when the problems are of the same size. In fact, when the approximation rank is too small, the method may even fail for less sparse and sensitive images. For this it is recommanded to increase the rank. But, this may increase the complexity or cause numerical instabilities.

Figure 6.4 presents three registrations of the crane image at level 6 (craneLev6) using FA4DMIR. It is observed that, by fixing the the rank to 4 (red line in left plot) improves the function reduction compared to the rank fixed to 2 (blue line in left plot) while it increases the compution time (red line in the right plot). However, if the rank is fixed to 6 (green line in both plots) one can observe that this does not improve the reduction of the function value and it does not reduce computing time. From this observation, one can note that it is hard to fix the rank in advance. In this

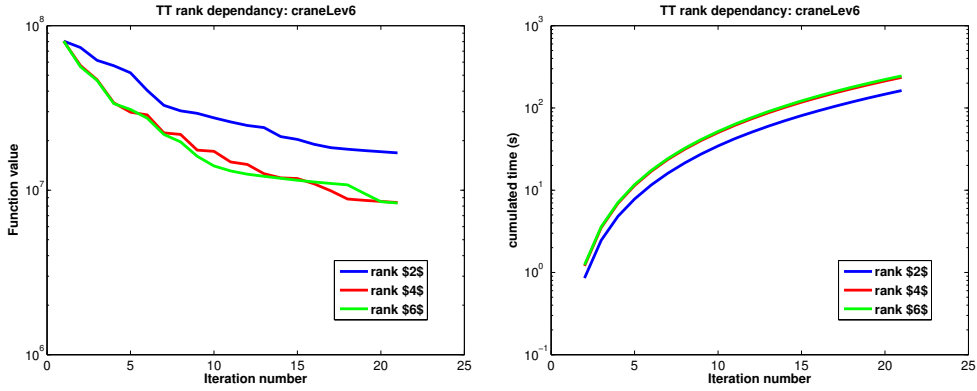


Figure 6.4: TT-rank dependency: Function value decreases more when the rank is increased (left plot) while this increases the computational time (right plot).

work the approximation rank were fixed taking into account the minimal value of the L-curve produced by the mlrankest tool from Tensorlab [50]. This L-curve allows to visualize a balance between the upperbound of different core tensors and the relative error of a given LMLRA [78]. With this plot, one gets insight about the interval where the rank should be picked. In our case, the minimal rank was often chosen.

Figure 6.5 visualizes the l-curve of the right-hand side of the foetus image (level 7, size $[2^7,\ 2^7,\ 2^6]$) and shows that the rank interval is $[3,\ 31]$. Thus, we fix our rank to 3.

6.2.2 PCG convergence: matrix format versus TT-format

The use of TT format is already a preconditioning techniques since this uses a low-rank approximation and allows more stable numerical operations. Figure 6.6 presents the reduction of the residual norm of the CG algorithm with respect to the

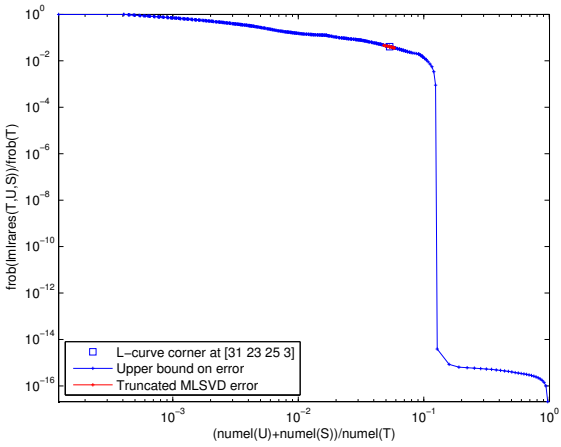


Figure 6.5: L-curve produced by the mlrankest from Tensorlab [50].

number of iterations. The aim is to compare the non-preconditioned matrix CG (red doted line) with the non-preconditioned TT CG(blue line) on the foetusLev6 (left) and foetusLev7 (right). It is observed that the CG in TT format, even with no preconditioner, is already a kind of preconditioned matrix CG. One can observe that for FoetusLev7, the rank 3 we picked is already improving the convergence rate.

However, it is important to note that, preconditioning in TT-format is very important, not only to accelerate the algorithms but also to stabilize the operations.

In the section below, we compare our algorithms on certain fixed levels of each of our images.

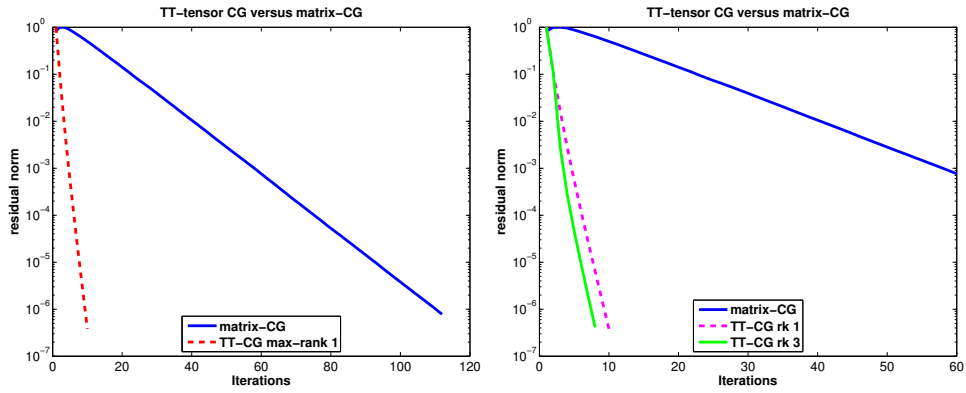


Figure 6.6: Matrix PCG convergence versus TT-PCG convergence on the FoeutusLev6 (left) and FoetusLev7 (right)

6.3 Comparison of algorithms at fixed levels

In what follows, we present the function value reduction and the cumulated time within 100 iterations for fixed levels of each image. The objectif is to point out the sensitivity of each of the three algorithms with respect to the increase of the problem size. This increase of the problem size is related to the size of the discretization step, that is smaller at high levels and larger for low levels. The goal is to highlight that, the FA4DMIR algorithm may be well suited for larger images. Some of our problems are presented below and others are in the Annexe F

In FAIR4DMIR, the TT-rank was fixed to 3 for all the problems. For FAIR and FAIR-GN we used Tchebychev polynomial preconditioner 4.4.3 and for FA4DMIR used the preconditioning via exponetial sums inverse 5.3.5.

6.3.1 The crane image

The 3D crane images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, service d'imagerie médicale. The images are of size [512, 512, 256] and they are registered on six levels as presented in the Table 6.1.

In this subsection, we analyze the registration on 4 levels: $4, 5, 6$ and 7 in the aim of comparison with other images.

Figure 6.7 presents the function value decrease with respect to the number of iterations on the crane images. It is observed that the FAIR algorithm (in blue) performs better for all most all the levels: level 4 (top left), level 5 (top right) and level 6 (bottom left). The FAIR-GN algorithm (in red) and the FA4DMIR algorithm (in green) are less attractive for these levels. However, one can observe that the gap is not very significant and the larger is the size of the image, better becomes the FA4DMIR algorithm compared to FAIR and FAIR-GN . It becomes even the most accurate at level 7 (bottom right). Although one should expect FAIR-GN to be the most attractive since it uses a kind of second order information, this is not the case for this image. The reason may be that, the Jacobian did not catch sufficient

Levels	Size	System size	Problem id
4	$(16, 16, 8)$	6 144	CraneLev4
5	$(32, 32, 16)$	49 152	CraneLev5
6	$(64, 64, 32)$	393 216	CraneLev6
7	$(128, 128, 64)$	3 145 728	CraneLev7
8	$(256, 256, 128)$	25 165 824	CraneLev8
9	$(512, 512, 256)$	201 326 592	CraneLev9

Table 6.1: Crane images: six levels of a couple of crane images in 3D. These images were provided by Hubert Meurisse, CHU-UCL-Namur, mont Godine hospital, nuclear medecine department. The images are of size [512, 512, 256]

second-order information. We now observe the time spent by each algorithm during the registration. The summary of reached values is presented 6.2

Figure 6.8 presents the cumulated computation time on the crane images. It is observed that FAIR is faster than FAIR-GN and FA4DMIR for low levels: level 4 (top left) and level 5 (top right) while for high level images FA4DMIR becomes faster: level 6 (bottom left) and level 7 (bottom right). This confirm that the FA4DMIR algorithm is faster and less sensitive to the problem size.

Table 6.2 summarizes the results reached by the algorithms presented in Figure 6.7 and Figure 6.8. Runing 100 iterations at each level, this table presents the reached function value (F_c), the ratio F_c/F_0 where F_c is the current function value and F_0 is the initial function value, and the cumulated time (T_c).

Figure 6.9 presents a sample of results for the crane image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

6.3.2 The brain image

The couple of $3D$ brain images is available in the FAIR package. According to notes in this package, these images were provided by Ron Kikinis, Surgical Planning Laboratory, Brigham and Women's Hospital, Boston. The images are of size [128, 64, 128] and were given on the domain $\Omega = [0, 20, 0, 10, 0, 20]$. They are registered on four levels.

Table 6.3 presents the size of the brain images at fixed levels and indicates the size of linear systems to be solved by iterative solvers at this level. In addition, an identification label is provided for each level to identify it as one of the problems in the database.

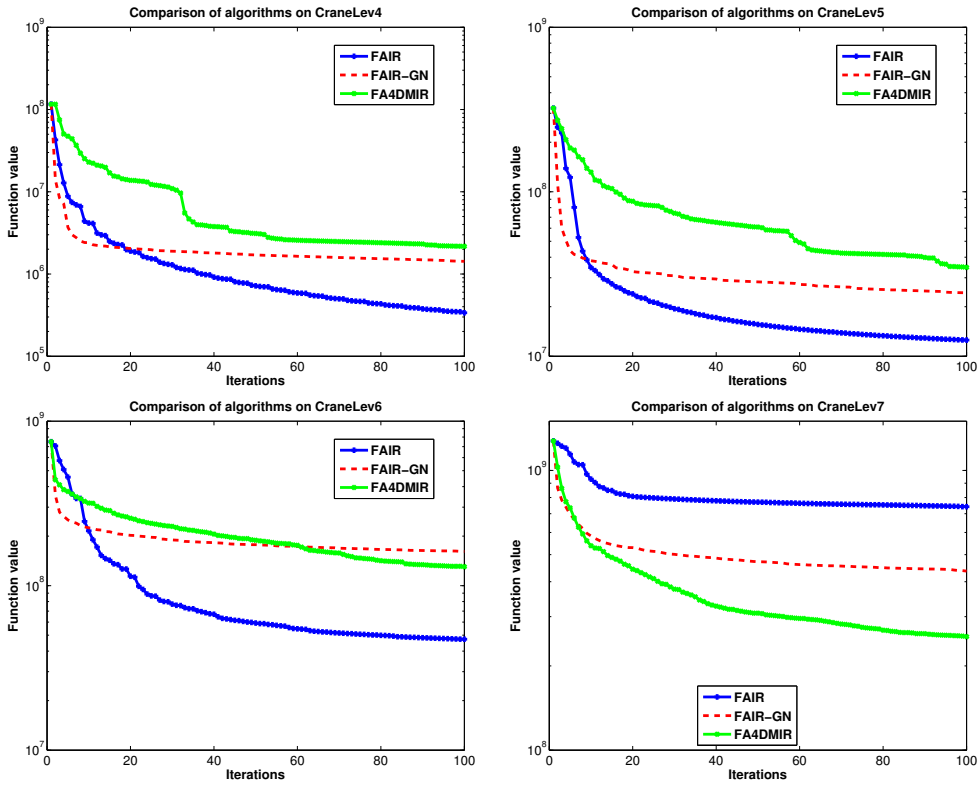


Figure 6.7: Crane images: comparison of the decrease of the function value with respect to the number of iterations for FAIR, FAIR-GN and FA4DMIR on crane image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (at bottom right).

Figure 6.10 presents the decrease of the function value with respect to the number of iterations for each of the three algorithms: FAIR (blue line), FAIR-GN (red doted line) and FA4DMIR (green line). It can be observed that The FAIR algorithm is the winner to lower levels (BrainLev4, at the top left, BrainLev5 at the top right and BrainLev6 at the bottom left) while the FAIR-GN algorithm performs well at level 7 (BrainLev7 at the bottom right). The FA4DMIR algorithm did not decrease sufficiently the function value for this image. However, one can observe that the FA4DMIR algorithm is less sensitive to the size of the problem. The behaviour of the FA4DMIR algorithm may be explained by the sensitivity of the brain images to truncation since they were acquired by Magnetic Resonance Images (MRI) 1.2. In fact, the sensitivity to truncation leads to lack of precision and thus the algorithm spends alot of time in the line search to find a step length that allows a suffiscient decrease. Thus, we guess that the less performance of the FA4DMIR algorithm on the brain images may be roughly explained by this sensitivity to truncation.

Figure 6.11 presents the cumulated computation time after 100 iterations. One

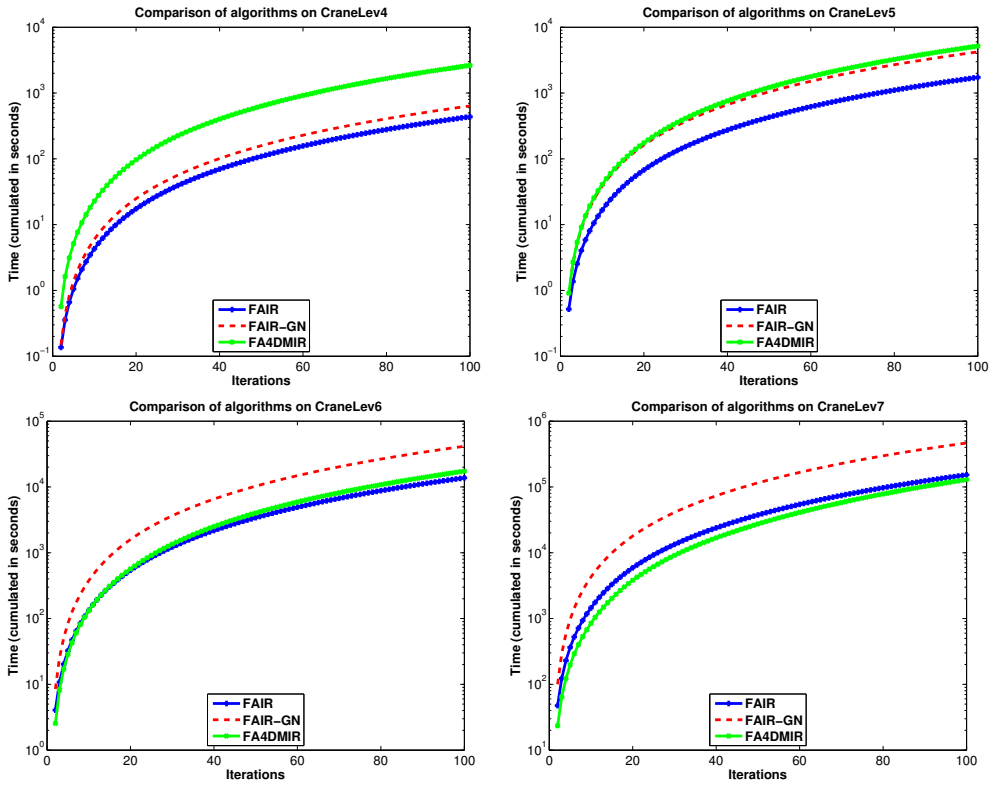


Figure 6.8: Crane images: comparison of cumulated computing time with respect to the number of iterations for FAIR, FAIR-GN and FA4DMIR on crane image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (at bottom right).

can see that FA4DMIR is the most expensive for the small level (BrainLev4 at the top left), but becomes faster when the size is enlarged: level 5 (top right), level 6 (bottom left) and level 7 (bottom right). Of course, this may of less importance since the FA4DMIR algorithm did not decrease the function sufficiently (see Figure 6.11). However, it is important to note that, since one may be interested in the best compromise between time and precision, the speed of FA4DMIR for larger images may be of importance, even for such images. A more concrete way to improve the rate of function decrease for FA4DMIR should be the increase of the maximal TT-rank.

Table 6.4 summarizes the results reached by the algorithms presented in figure Figure 6.10 and Figure 6.11. This table presents the reached function value (F_c), the ratio F_c/F_0 where F_c is the current function value and F_0 is the initial function value, and the cumulated time (T_c).

The less effectiveness of the FA4DMIR algorithm for this image can be explained by the nature of the image. First, the brain image was dense (no zero entry) and

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$1.1\,10^{8}$	1	0
FAIR	$5.3\,10^{5}$	$4.8\,10^{-3}$	$4.4\,10^{2}$
FAIR-GN	$1.4\,10^{6}$	$1.2\,10^{-2}$	$6.54\,10^{2}$
FA4DMIR	$4.1\,10^{6}$	$3.7\,10^{-2}$	$2.6\,10^{3}$
Level 5			
From	$5.2\,10^{8}$	1	0
FAIR	$1.2\,10^{7}$	$2.3\,10^{-2}$	$1.7\,10^{3}$
FAIR-GN	$5.4\,10^{7}$	$1.0\,10^{-1}$	$4.3\,10^{3}$
FA4DMIR	$6.4\,10^{7}$	$1.2\,10^{-1}$	$5.2\,10^{3}$
Level 6			
From	$8.5\,10^{8}$	1	0
FAIR	$7.7\,10^{7}$	$9.0\,10^{-2}$	$5.6\,10^{3}$
FAIR-GN	$2.6\,10^{8}$	$3.0\,10^{-1}$	$1.2\,10^{4}$
FA4DMIR	$2.2\,10^{8}$	$2.5\,10^{-1}$	$5.1\,10^{3}$
Level 7			
From	$2.2\,10^{9}$	1	0
FAIR	$9.4\,10^{8}$	$4.2\,10^{-1}$	$8.9\,10^{4}$
FAIR-GN	$6.3\,10^{8}$	$2.8\,10^{-1}$	$3.5\,10^{5}$
FA4DMIR	$5.5\,10^{8}$	$2.5\,10^{-1}$	$8.4\,10^{5}$

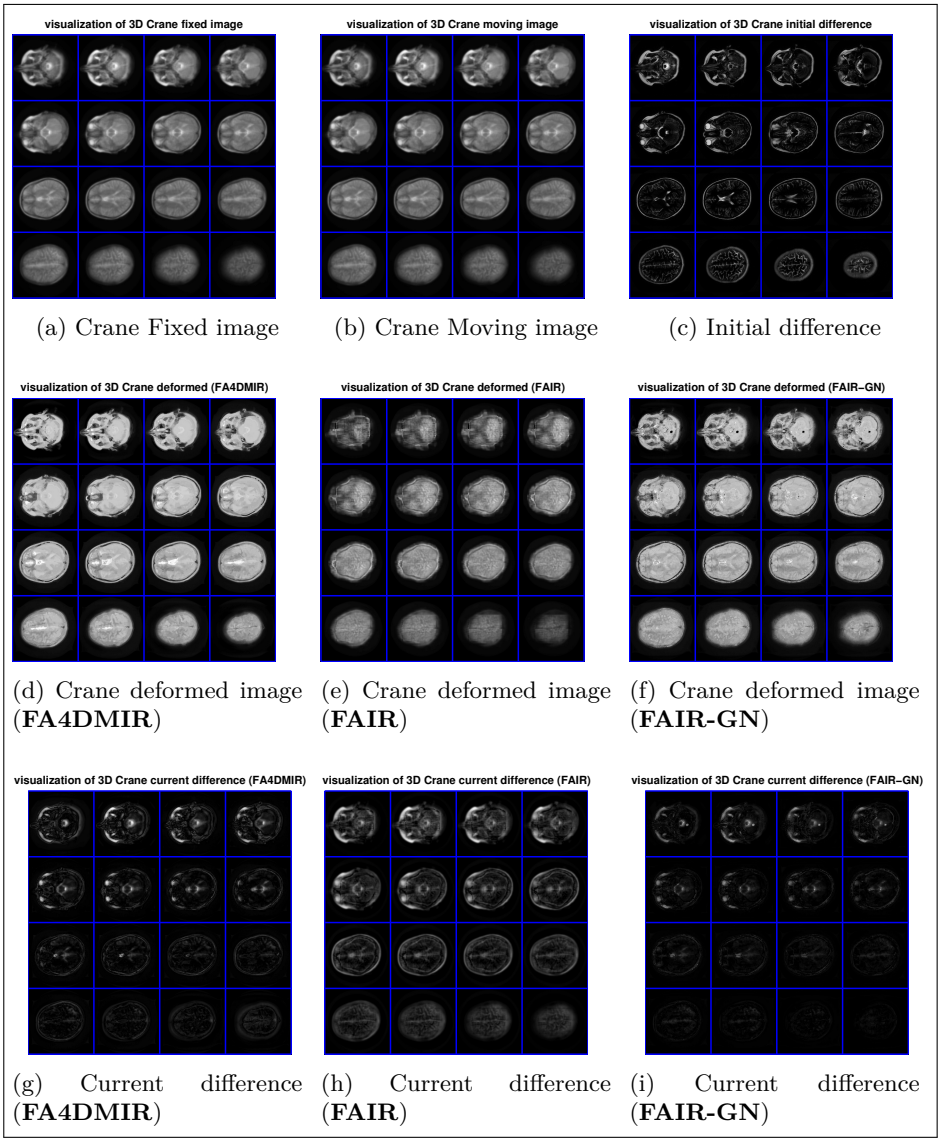
Table 6.2: Crane images: for each level of the crane images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time time (T_c) for 100 iterations.

Levels	Size	System size	Problem id
4	$(16, 8, 16)$	6 144	BrainLev4
5	$(32, 16, 32)$	49 152	BrainLev5
6	$(64, 32, 64)$	393 216	BrainLev6
7	$(128, 64, 128)$	3 145 728	BrainLev7

Table 6.3: Brain images: four levels of a couple of brain images in 3D. These images are available in the FAIR package [10].

so less sparse than the others. Second, we argue that, since the brain image is acquired by MRI techniques (see 1.2) it may be more sensitive to truncation based on singular value decomposition since the acquisition takes into account the orientations of the excited nuclei represented in some singular vectors associated to very small eigenvalues. So, much truncation may lead to loss of significant information when related singular values are put to zero.

Figure 6.12 presents a sample of results for the brain image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row:



(a) Crane Fixed image (b) Crane Moving image (c) Initial difference

(d) Crane deformed image (**FA4DMIR**) (e) Crane deformed image (**FAIR**) (f) Crane deformed image (**FAIR-GN**)

(g) Current difference (**FA4DMIR**) (h) Current difference (**FAIR**) (i) Current difference (**FAIR-GN**)

Figure 6.9: Crane images: sample of crane image registration results at level 7.

fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

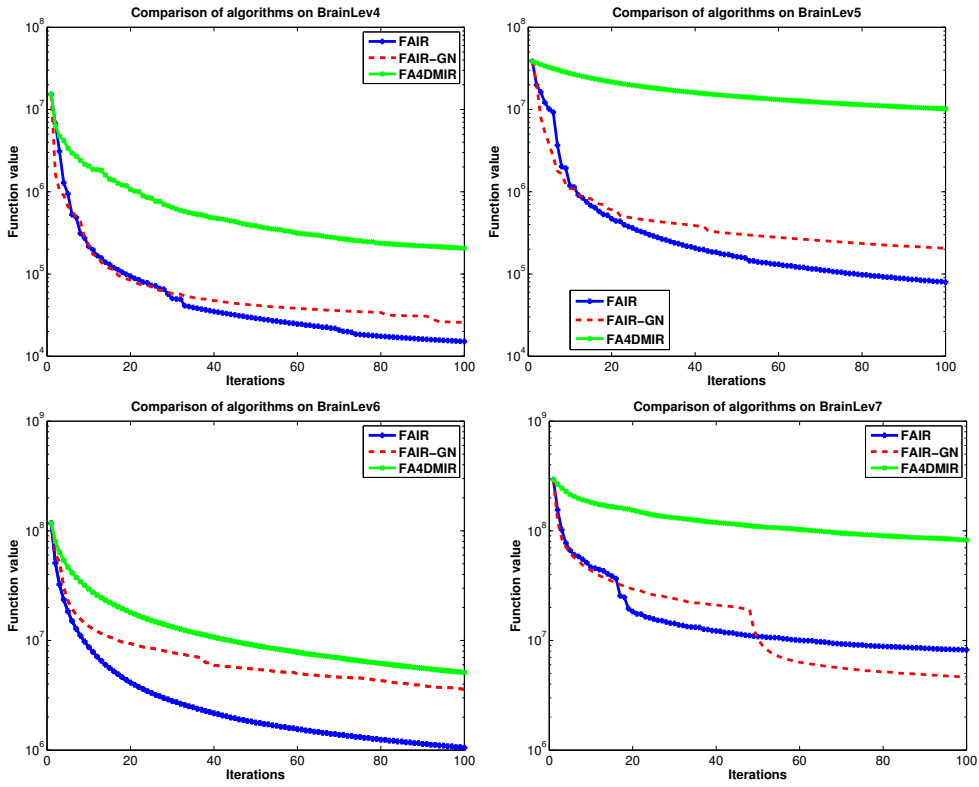


Figure 6.10: Brain images: comparison of the decrease of the function value with respect to the number of iterations for FAIR (blue), FAIR-GN (red) and FA4DMIR (green) on brain image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

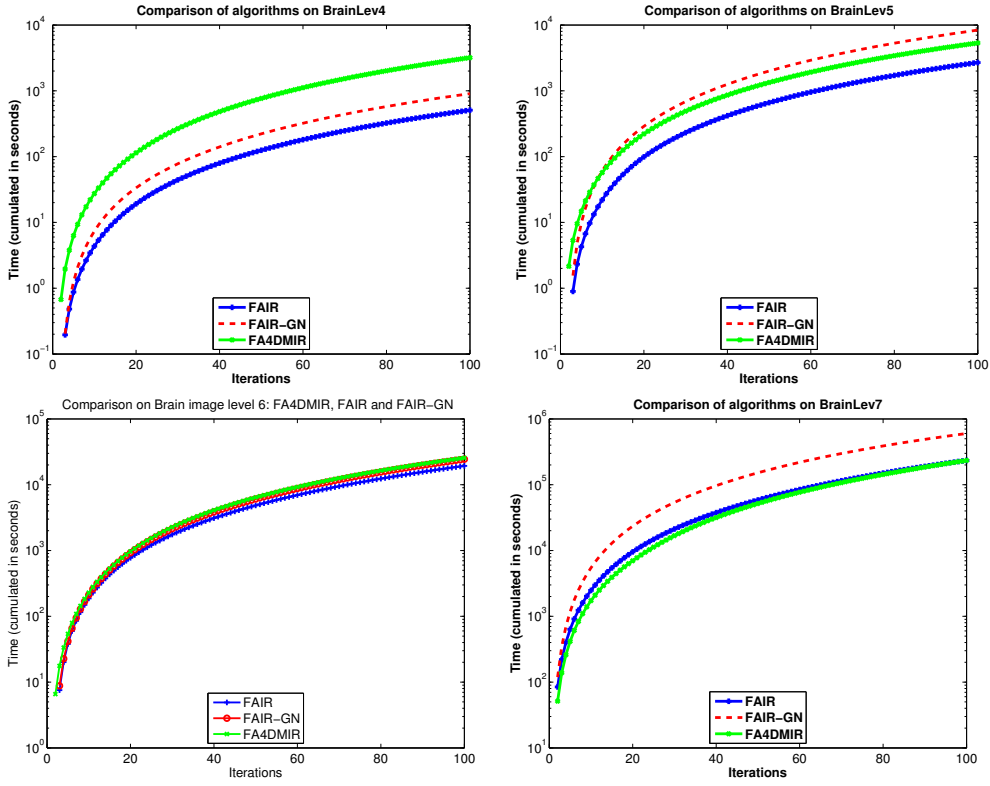
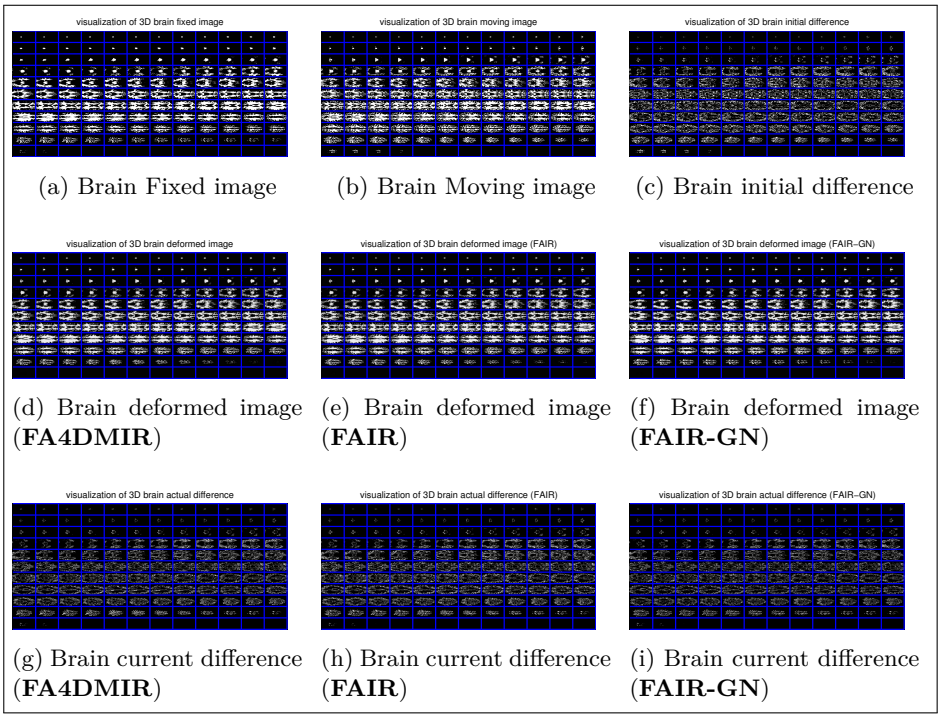


Figure 6.11: Brain images: comparison of the cumulated time with respect to the number of iterations for FAIR (blue line), FAIR-GN (red line) and FA4DMIR (green line) on brain image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$1.5\ 10^{7}$	1	0
FAIR	$1.5\,10^{4}$	$1.0\,10^{-3}$	$1.5\,10^{2}$
FAIR-GN	$2.5\,10^{4}$	$1.6\,10^{-3}$	$4.1\,10^{2}$
FA4DMIR	$2.8\,10^{5}$	$1.8\,10^{-1}$	$1.2\,10^{3}$
Level 5			
From	$5.8\ 10^{7}$	1	0
FAIR	$9.8\,10^{4}$	$1.6\,10^{-3}$	$2.7\,10^{3}$
FAIR-GN	$2.9\,10^{5}$	$5.0\,10^{-3}$	$8.7\,10^{3}$
FA4DMIR	$1.01\,10^{7}$	$1.7\,10^{-1}$	$6.4\,10^{3}$
Level 6			
From	$1.1\ 10^{8}$	1	0
FAIR	$1.05\,10^{6}$	$9.5\,10^{-3}$	$2.2\,10^{4}$
FAIR-GN	$7.05\,10^{6}$	$6.4\,10^{-2}$	$2.2\,10^{4}$
FA4DMIR	$9.36\,10^{6}$	$8.5\,10^{-2}$	$2.6\,10^{4}$
Level 7			
From	$2.9\ 10^{8}$	1	0
FAIR	$8.19\,10^{6}$	$2.8\,10^{-2}$	$2.38\,10^{5}$
FAIR-GN	$4.62\,10^{6}$	$1.5\,10^{-2}$	$6.13\,10^{5}$
FA4DMIR	$9.8\,10^{7}$	$3.3\,10^{-1}$	$2.39\,10^{5}$

Table 6.4: Brain images: for each level of the brain images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).



(a) Brain Fixed image (b) Brain Moving image (c) Brain initial difference

(d) Brain deformed image (**FA4DMIR**) (e) Brain deformed image (**FAIR**) (f) Brain deformed image (**FAIR-GN**)

(g) Brain current difference (**FA4DMIR**) (h) Brain current difference (**FAIR**) (i) Brain current difference (**FAIR-GN**)

Figure 6.12: Brain image registration results: level 7

6.3.3 The foetus images

The $3D$ foetus images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size [512, 512, 128] and they are registered on five levels as presented in the Table 6.5.

Levels	Size	System size	Problem id
4	$(32, 32, 8)$	24 576	FoetusLev4
5	$(64, 64, 16)$	196 608	FoetusLev5
6	$(128, 128, 32)$	1 572 864	FoetusLev6
7	$(256, 256, 64)$	12 582 912	FoetusLev7
8	$(512, 512, 128)$	100 663 296	FoetusLev8

Table 6.5: Foetus images: five levels of a couple of foetus images in 3D. These images were provided by Hubert Meurisse, CHU-UCL-Namur, mont Godine hospital, nuclear medecine department.

We analyse, here again, the registration on 4 fixed levels: $4, 5, 6$ and 7 in the aim of comparison. while the level 8 is shown in Annexe F.

Figure 6.13 presents the decrease of the function value on the foetus images with respect to the number of iterations. Once more, in the top plots (level 4 and level 5) the FAIR algorithm (in blue) performs better than FAIR-GN (in red) and FA4DMIR (in green) However, in the bottom plots (level 6 and level 7) the preferences between FAIR and FA4DMIR algorithms changes. Now, it is observed that FA4DMIR (in green) performs better. Since the high levels are more expensive that lower levels, the algorithm that performs well at high levels is to be preferred. Thus, FA4DMIR is the winner here. Observe now the computing time.

Figure 6.14 presents the cumulated computation time for foetus images. Once more, it is observed that FAIR is faster than FAIR-GN and FA4DMIR for the top plots (levels 4 and 5) and recall that it decreases better the function value for those levels. These are relatively small images. In the bottom left plot (level 6) the FA4DMIR algorithm becomes as faster as the FAIR algorithm. In the bottom right plot (level 7) the FA4DMIR algorithm is now faster than FAIR while the FAIR-GN stills the most expensive as expected. The fact that foetus images were acquired by CT methods explains the improvements of the FA4DMIR algorithm.

Table 6.6 summarizes the results reached by the algorithms presented in Figure 6.13 and Figure 6.14.

Figure 6.15 presents a sample of results for the foetus image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

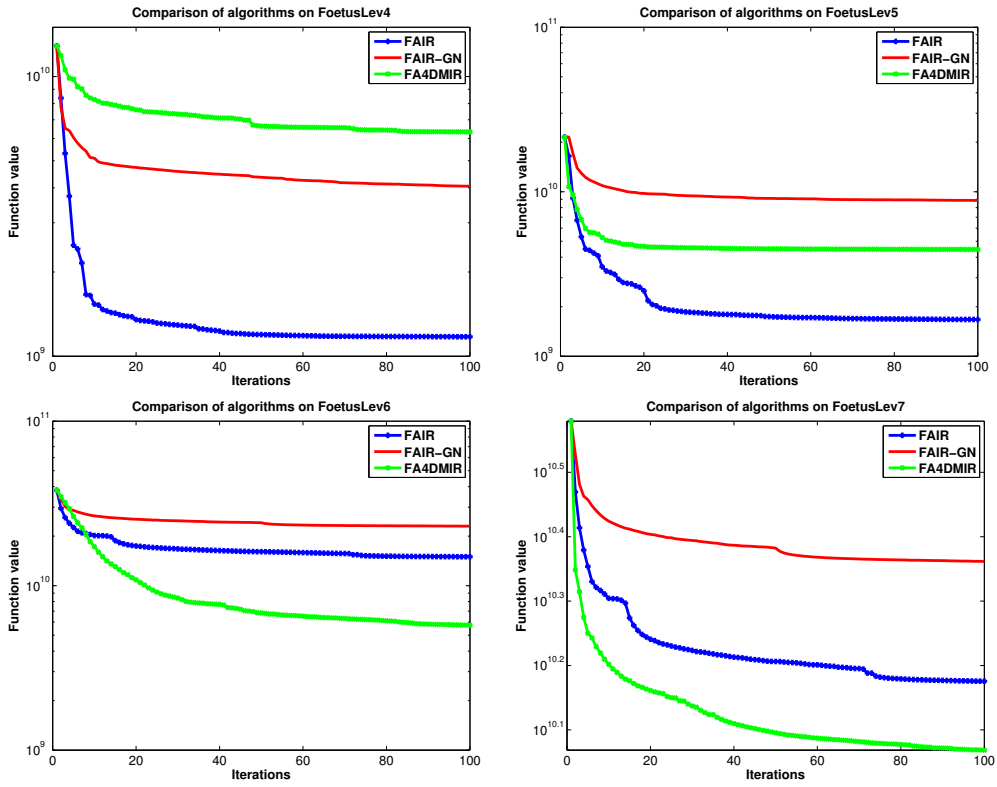


Figure 6.13: Foetus images: Comparison of the function value decrease with respect to the number of iterations for FAIR (blue line), FAIR-GN (red line) and FA4DMIR (green line) on brain image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

6.3.4 The mice image

The $3D$ mice images here are available in the FAIR package [10]. The images are of size [128, 128, 32] and were given on the domain $\Omega = [0, 128, 0, 128, 0, 32]$. They are registered on three levels: $4, 5$ and 6 as presented in the Table 6.7.

Figure 6.16 presents the decrease of the function value with respect to the number of iterations on the mice images registration. Contrary to expectations, it is observed that the FA4DMIR algorithm (in green) performs better than FAIR algorithm and FAIR-GN algorithm at all levels (from level 4 to level 6). These are small images. The main explication is that the mice images are sparse. The original images were of size $[128, 128, 24]$ what means some zeros slices have been added and this makes the problem sparse.

Figure 6.17 presents the cumulated computation time after 100 iterations on the mice images. One can see that, although FA4DMIR decreased significantly the function, it is expensive for level 4 (top left) and for level 5 (top right) than the FAIR

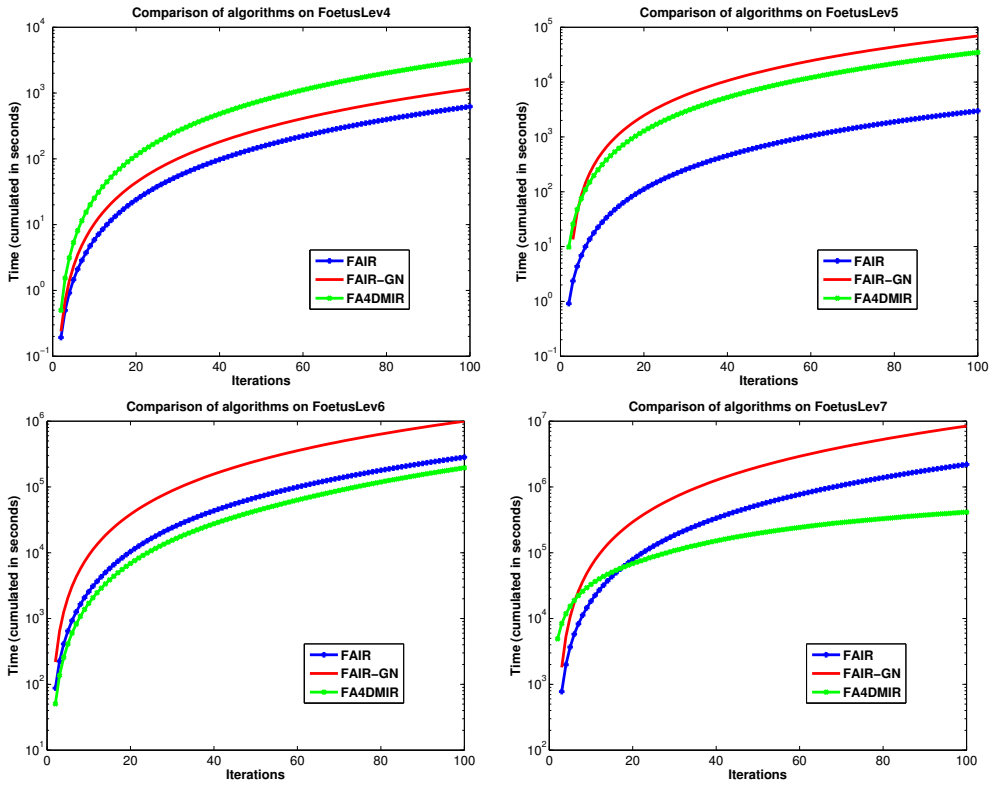


Figure 6.14: Foetus images: Comparison of the cumulated time with respect to the number of iterations for FAIR (blue line), FAIR-GN (red line) and FA4DMIR (green line) on brain image registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

algorithm. It becomes as faster as FAIR at level 6 (bottom plot). This is quite normal since these images are of small size, and thus the other methods perform well.

Table 6.8 summarizes the results reached by the algorithms presented in figure Figure 6.16 and Figure 6.17. Runing 100 iterations for each level on the mice images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).

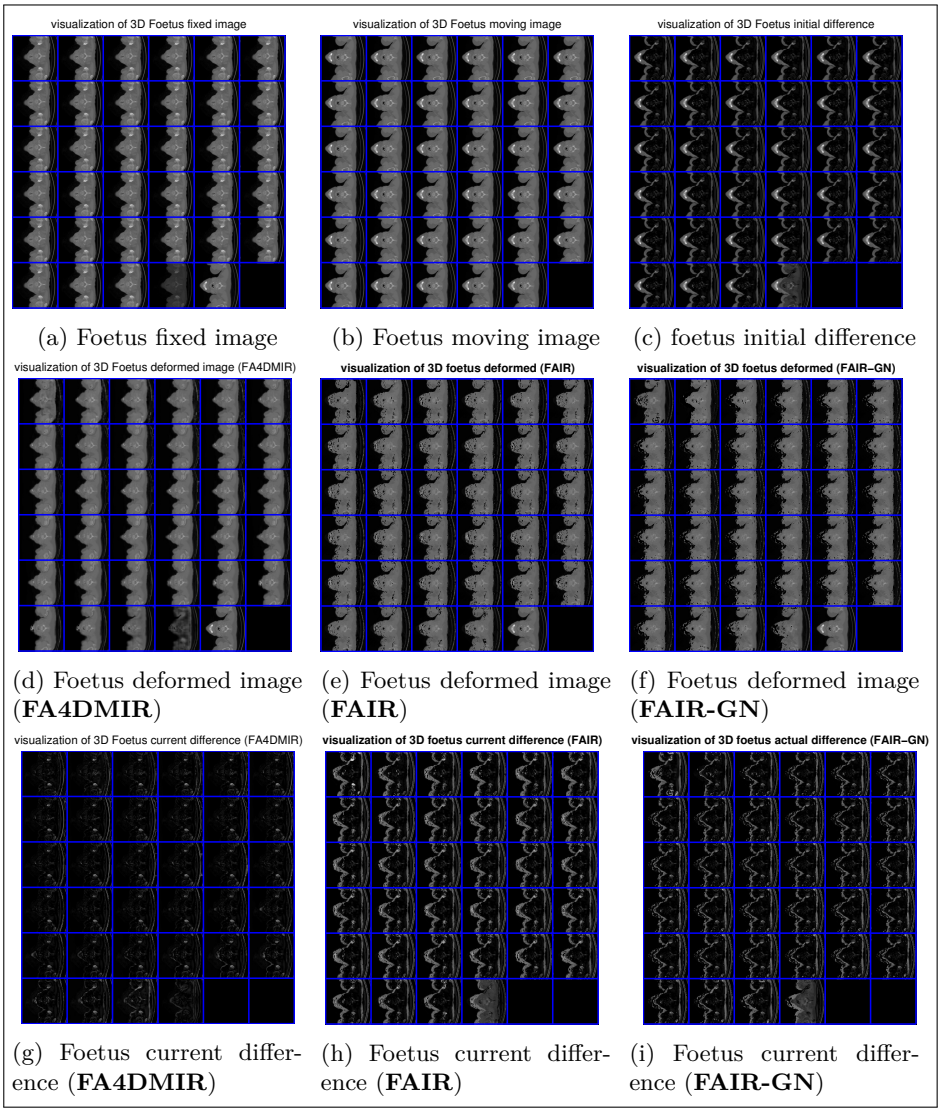
Figure 6.18 presents a sample of results for the mice image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$2.2\,10^{10}$	1	0
FAIR	$1.2\,10^{9}$	$5.4\,10^{-2}$	$2.3\,10^{2}$
FAIR-GN	$6.0\,10^{9}$	$2.7\,10^{-1}$	$3.1\,10^{2}$
FA4DMIR	$7.3\,10^{9}$	$3.3\,10^{-1}$	$1.2\,10^{3}$
Level 5			
From	$4.1\ 10^{10}$	1	0
FAIR	$2.6\,10^{9}$	$6.3\,10^{-2}$	$8.9\,10^{2}$
FAIR-GN	$1.8\,10^{10}$	$4.3\,10^{-1}$	$8.5\,10^{4}$
FA4DMIR	$6.8\,10^{9}$	$1.6\,10^{-1}$	$1.2\,10^{4}$
Level 6			
From	$5.9\ 10^{10}$	1	0
FAIR	$2.4\,10^{10}$	$4.0\,10^{-1}$	$3.2\,10^{5}$
FAIR-GN	$5.1\,10^{10}$	$8.6\,10^{-1}$	$1.0\,10^{6}$
FA4DMIR	$7.9\,10^{9}$	$1.3\,10^{-1}$	$1.2\,10^{5}$
Level 7			
From	$3.9\ 10^{10}$	1	0
FAIR	$1.7\,10^{10}$	$4.3\,10^{-1}$	$9.8\,10^{5}$
FAIR-GN	$2.5\,10^{10}$	$6.4\,10^{-1}$	$9.7\,10^{6}$
FA4DMIR	$1.0\,10^{10}$	$3.9\,10^{-1}$	$1.2\,10^{5}$

Table 6.6: Foetus images: for each level of the foetus images, this table presents the reached functional value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).

Levels	Size	System size	Problem id
4	$(32, 32, 8)$	24 576	MiceLev4
5	$(64, 64, 16)$	196 608	MiceLev5
6	$(128, 128, 32)$	1 572 864	MiceLev6

Table 6.7: Mice: three levels of a couple of mice images in 3D. These images are available in the FAIR package from Jan Modersitzki.



(a) Foetus fixed image (b) Foetus moving image (c) foetus initial difference

(d) Foetus deformed image (**FA4DMIR**) (e) Foetus deformed image (**FAIR**) (f) Foetus deformed image (**FAIR-GN**)

(g) Foetus current difference (**FA4DMIR**) (h) Foetus current difference (**FAIR**) (i) Foetus current difference (**FAIR-GN**)

Figure 6.15: Foetus image registration results (sample)

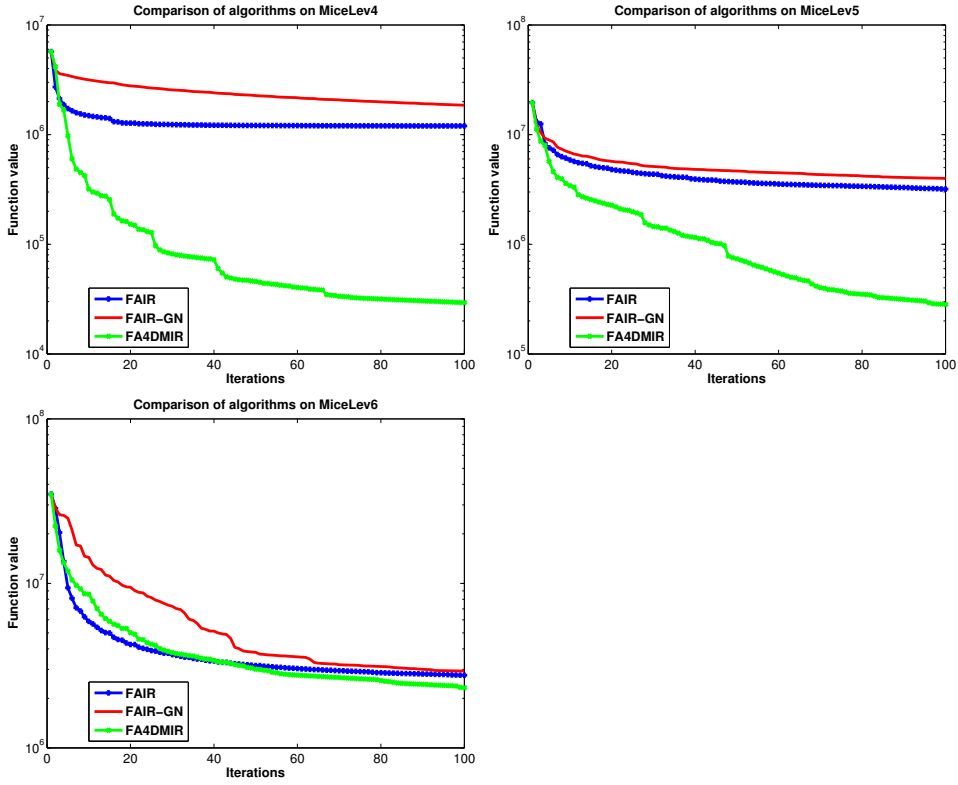


Figure 6.16: Mice images: comparison of the function value decrease with respect to the number of iterations on 3D mice images. level 4 (top left), level 5 (top right) and level 6 (bottom)

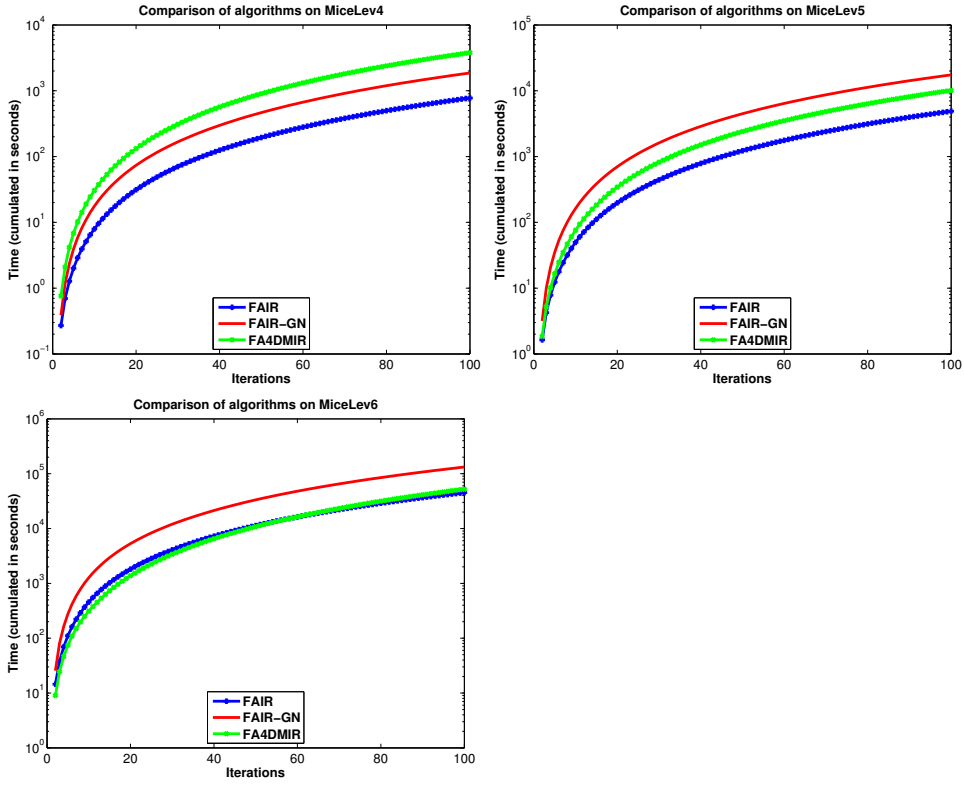
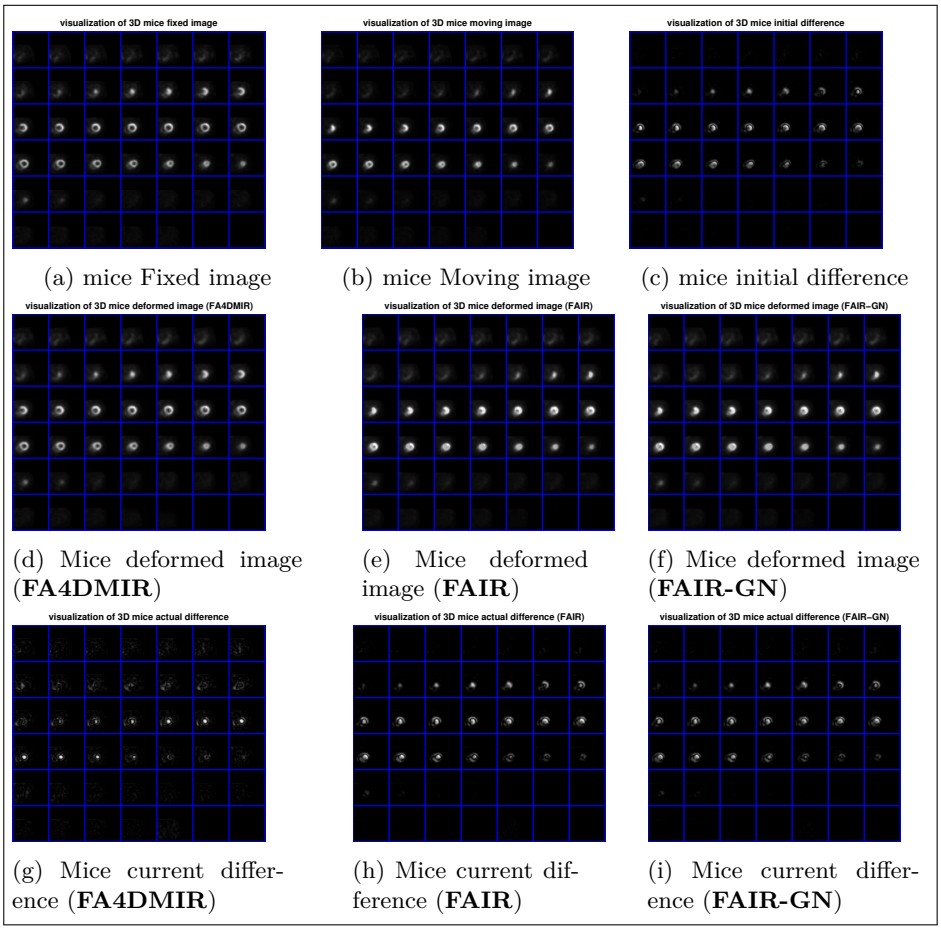


Figure 6.17: Mice images: comparison of computation time with respect to the number of iterations for mice images.level 4 (top left), level 5 (top right) and level 6 (bottom)

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$5.7\,10^{6}$	1	0
FAIR	$1.1\,10^{6}$	$1.9\,10^{-1}$	$7.9\,10^{2}$
FAIR-GN	$1.8\,10^{6}$	$3.1\,10^{-1}$	$1.9\,10^{3}$
FA4DMIR	$2.9\,10^{4}$	$5.0\,10^{-3}$	$3.8\,10^{3}$
Level 5			
From	$1.9\ 10^{7}$	1	0
FAIR	$3.1\,10^{6}$	$1.6\,10^{-1}$	$4.9\,10^{3}$
FAIR-GN	$3.9\,10^{6}$	$2.0\,10^{-1}$	$1.7\,10^{4}$
FA4DMIR	$2.8\,10^{5}$	$1.4\,10^{-2}$	$1.0\,10^{4}$
Level 6			
From	$3.4\ 10^{7}$	1	0
FAIR	$2.7\,10^{6}$	$7.9\,10^{-2}$	$4.6\,10^{4}$
FAIR-GN	$2.9\,10^{6}$	$8.3\,10^{-2}$	$1.3\,10^{5}$
FA4DMIR	$2.3\,10^{6}$	$6.7\,10^{-2}$	$5.3\,10^{4}$

Table 6.8: Mice images: for each level, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time time (T_c) after 100 iterations on mice images.



(a) mice Fixed image (b) mice Moving image (c) mice initial difference

(d) Mice deformed image (**FA4DMIR**) (e) Mice deformed image (**FAIR**) (f) Mice deformed image (**FAIR-GN**)

(g) Mice current difference (**FA4DMIR**) (h) Mice current difference (**FAIR**) (i) Mice current difference (**FAIR-GN**)

Figure 6.18: Mice image registration results

6.3.5 The knee image

The $3D$ knee images are available in the FAIR package and were provided by Thomas Netsch, philips research, Hambourg, Germany. The images of size [128, 64, 128] were given on the domain $\Omega = [0, 128, 0, 62, 0, 128]$ and are registered on four levels as shown in Table 6.9

Level	Size	System size	Problem id
4	$(16, 8, 16)$	$6\,144 \times 6\,144$	KneeLev4
5	$(32, 16, 32)$	$49\,152 \times 49\,152$	KneeLev5
6	$(64, 32, 64)$	$393\,216 \times 393\,216$	KneeLev6
7	$(128, 64, 128)$	$3\,145\,728 \times 3\,145\,728$	KneeLev7

Table 6.9: Knee images: four levels of a couple of knee images in 3D. These images are available in the FAIR package.

Figure 6.19 presents the decrease of the function value with respect to the number of iterations on the knee images registration. For these images, the FAIR-GN algorithm (in red) performs better at all levels (level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right)) than the FAIR algorithm (in blue) and FA4DMIR algorithm (in green). Comparison between FAIR and FA4DMIR shows that the FA4DMIR algorithm perform well than the FAIR algorithm even for low levels (small size images). This may be explained by the nature of the image. In fact, the knee image seems to be less sensitive to truncation since they were acquired by CT-scan techniques . Thus, good local approximations allow faster registration for FAIR-GN and less sensitivity explains the performance of FA4DMIR compared to FAIR.

Figure 6.20 presents the cumulated computation time after 100 iterations. It is observed that FA4DMIR is faster than FAIR and FAIR-GN at all the levels (level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right)). This confirm that these images are less sensitive to truncation. For these images, the algorithms behave as expected. The FAIR-GN algorithm is the most precise but the most expensive while the FA4DMIR is the faster.

Table 6.10 summarizes the results reached by the algorithms presented in figure Figure 6.19 and Figure 6.20 . Runing 100 iterations for each level on the knee images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).

Figure 6.21 presents a sample of results for the knee image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

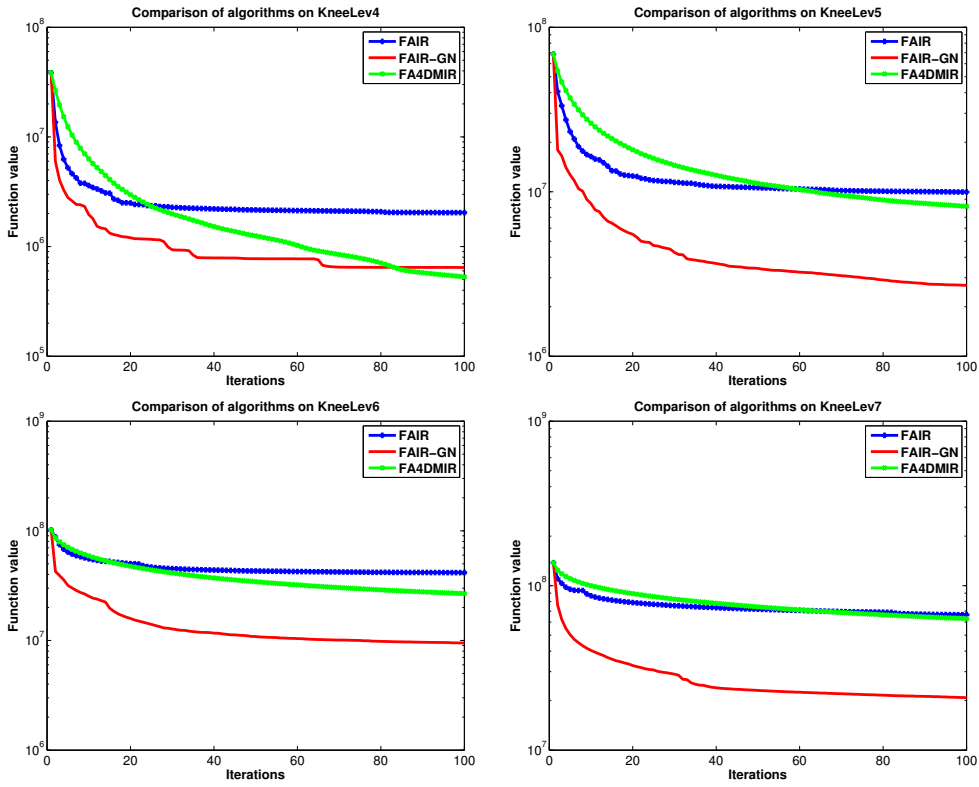


Figure 6.19: Knee images: comparison of the function value with respect to the number of iterations on the knee images registration at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

6.3.6 The chest image

The $3D$ chest images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size [512, 512, 1024] and they are registered on six levels as presented in the Table 6.11.

In this subsection, we analyze the registration on 4 levels: $4, 5, 6$ and 7 in the aim of comparison with other images.

Figure 6.22 presents the decrease of the function value with respect to the number of iterations on the chest image registration. It can be observed that, the FAIR-GN algorithm (in red) performs better at low levels (level 4 (top left) and level 5 (top right)) while FA4DMIR algorithm (in red) is better at high levels (level 6 at the bottom left and level 7 at the bottom right). Observe that even for low levels (4 and 5), the FA4DMIR algorithm performs well than the FAIR algorithm everywhere. Once again, the reason may be the nature of the image (CT-scan) but not the sparsity patterns because these image where dense in average.

Figure 6.23 presents the cumulated computation time after 100 iterations on the

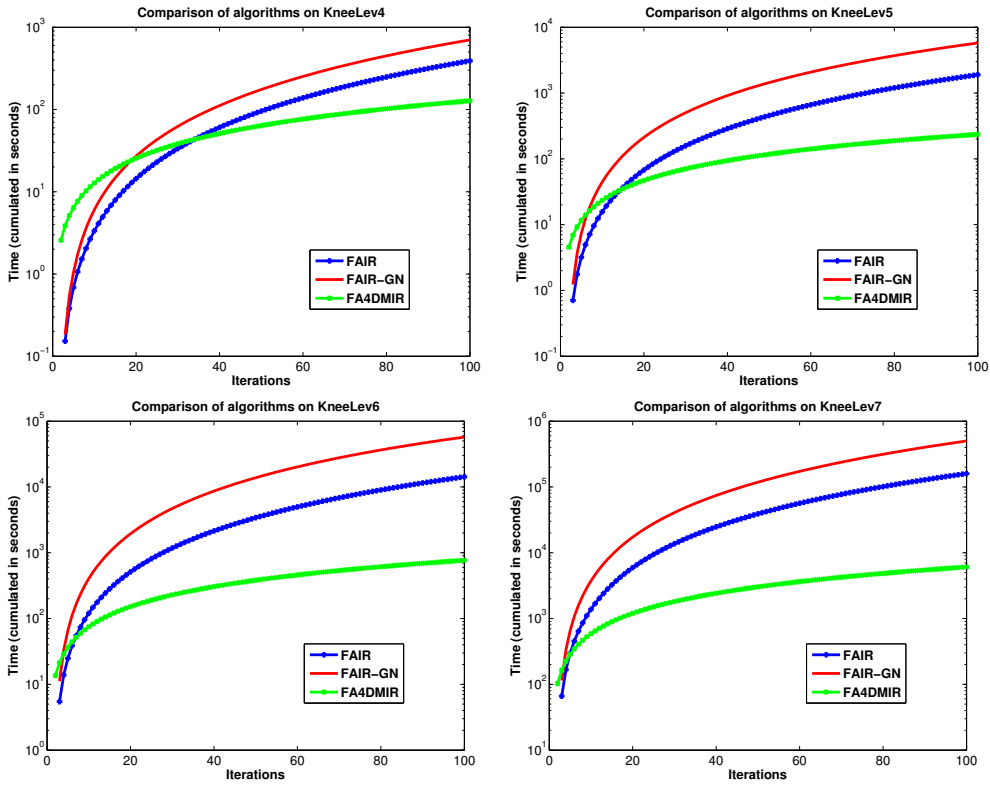


Figure 6.20: Knee images: comparison of FAIR (blue), FAIR-GN (red) and FA4DMIR (green) with respect of the computation time at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

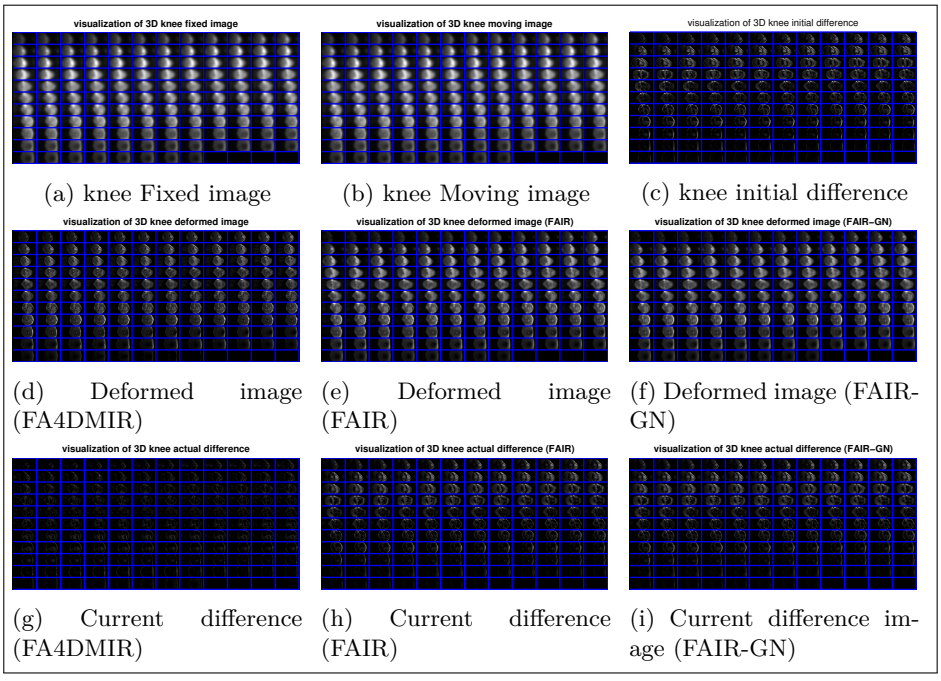
chest images. It is observed that FAIR-GN is as faster as FA4DMIR for low levels: level 4 (top left) and level 5 (top right) but this becomes faster for high levels: level 6 (bottom left) and level 7 (bottom right). Comparison between FAIR and FA4DMIR shows that this last performs better for any level.

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$5.8\,10^{7}$	1	0
FAIR	$3.2\,10^{6}$	$5.5\,10^{-2}$	$5.1\,10^{2}$
FAIR-GN	$9.8\,10^{5}$	$1.6\,10^{-2}$	$3.1\,10^{2}$
FA4DMIR	$8.9\,10^{5}$	$1.5\,10^{-2}$	$8.7\,10^{1}$
Level 5			
From	$6.8\,10^{7}$	1	0
FAIR	$9.9\,10^{6}$	$1.4\,10^{-2}$	$4.9\,10^{3}$
FAIR-GN	$2.6\,10^{6}$	$3.8\,10^{-3}$	$1.1\,10^{3}$
FA4DMIR	$1.9\,10^{7}$	$2.7\,10^{-1}$	$1.2\,10^{2}$
Level 6			
From	$1.0\,10^{8}$	1	0
FAIR	$8.1\,10^{7}$	$8.1\,10^{-1}$	$1.4\,10^{4}$
FAIR-GN	$9.4\,10^{6}$	$9.4\,10^{-2}$	$5.9\,10^{4}$
FA4DMIR	$4.1\,10^{7}$	$4.1\,10^{-1}$	$3.0\,10^{2}$
Level 7			
From	$1.3\,10^{8}$	1	0
FAIR	$6.6\,10^{7}$	$5.0\,10^{-1}$	$5.8\,10^{4}$
FAIR-GN	$2.0\,10^{7}$	$1.5\,10^{-1}$	$1.6\,10^{5}$
FA4DMIR	$6.0\,10^{7}$	$4.6\,10^{-1}$	$3.2\,10^{3}$

Table 6.10: Knee images: for each level of the knee images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.

Levels	Size	System size	Problem id
4	$(16, 16, 32)$	24 576	ChestLev4
5	$(32, 32, 64)$	196 608	ChestLev5
6	$(64, 64, 128)$	1 572 864	ChestLev6
7	$(128, 128, 256)$	12 582 912	ChestLev7
8	$(256, 256, 1024)$	100 663 296	ChestLev8
9	$(512, 512, 1024)$	805 306 368	ChestLev9

Table 6.11: Chest images: six levels of a couple of chest images in 3D. These images were provided by Hubert Meurisse, CHU-UCL-Unamur mont Godine hospital. Nuclear medicine



(a) knee Fixed image (b) knee Moving image (c) knee initial difference

(d) Deformed image (FA4DMIR) (e) Deformed image (FAIR) (f) Deformed image (FAIR-GN)

(g) Current difference (FA4DMIR) (h) Current difference (FAIR) (i) Current difference image (FAIR-GN)

Figure 6.21: Knee image registration results

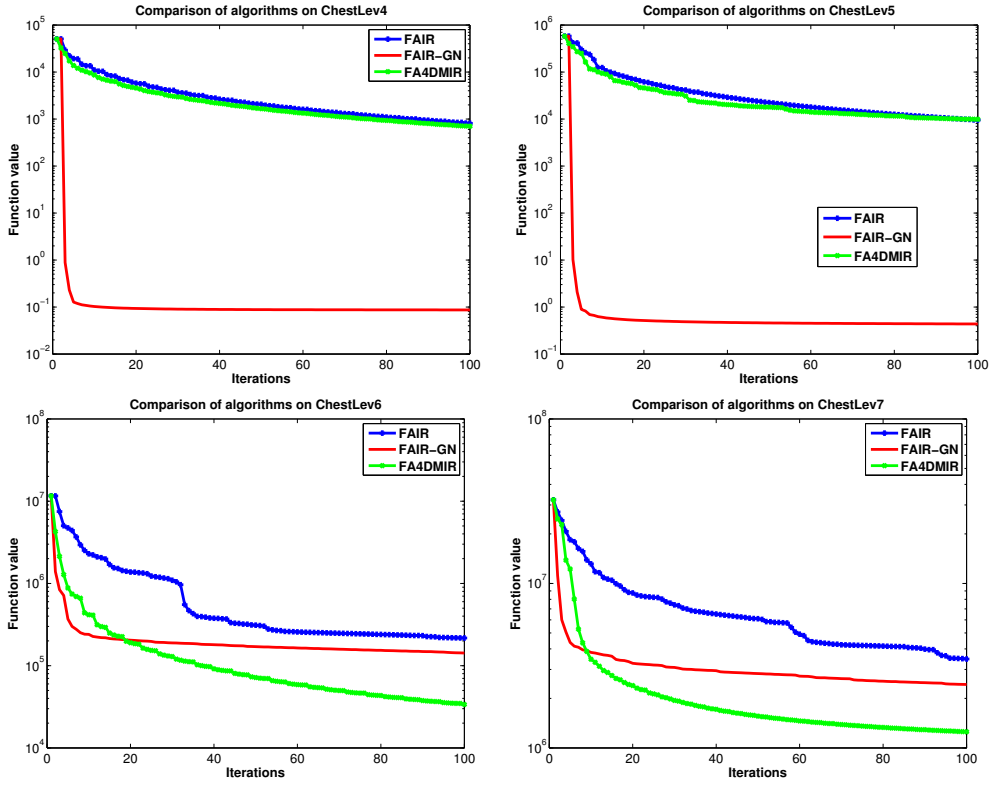


Figure 6.22: Chest images: Comparison of the decrease of the function value with respect to the number of iterations for FAIR (blue line), FAIR-GN (red line) and FA4DMIR (green line) at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (at bottom right).

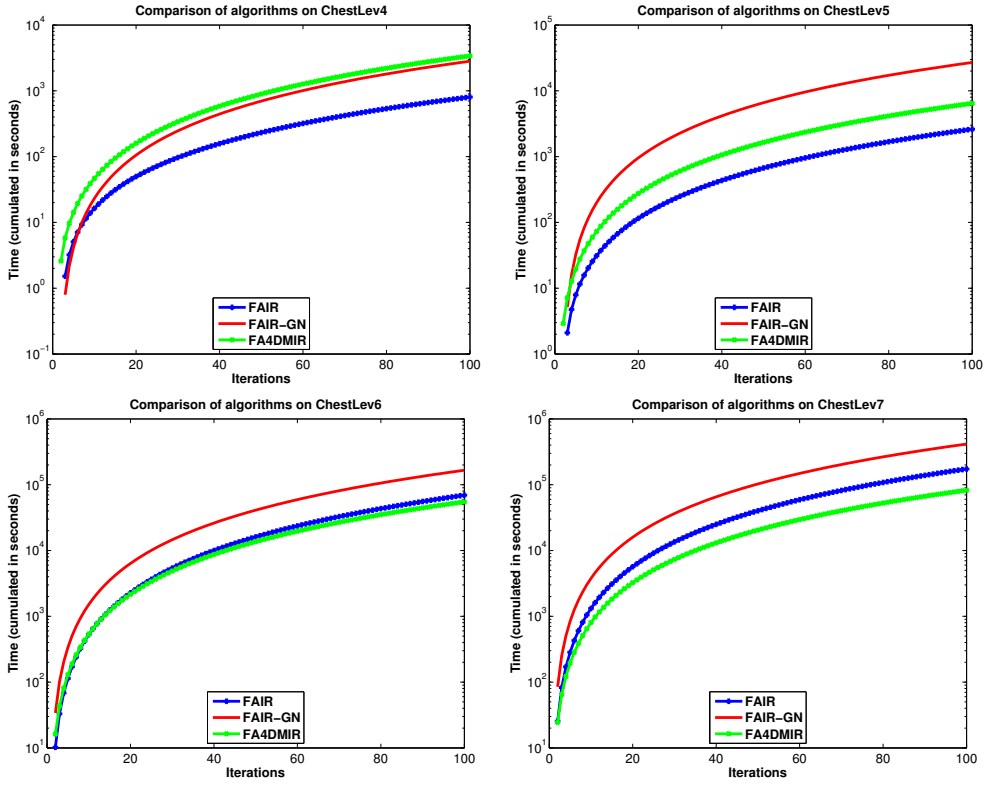


Figure 6.23: Chest images: comparison of the cumulated computation time with respect to the number of iterations for FAIR (blue line), FAIR-GN (red line) and FA4DMIR (green line) at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

Table 6.12 summarizes the results reached by the algorithms presented in figure Figure 6.22 and Figure 6.23 . Runing 100 iterations for each level on the chest images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).

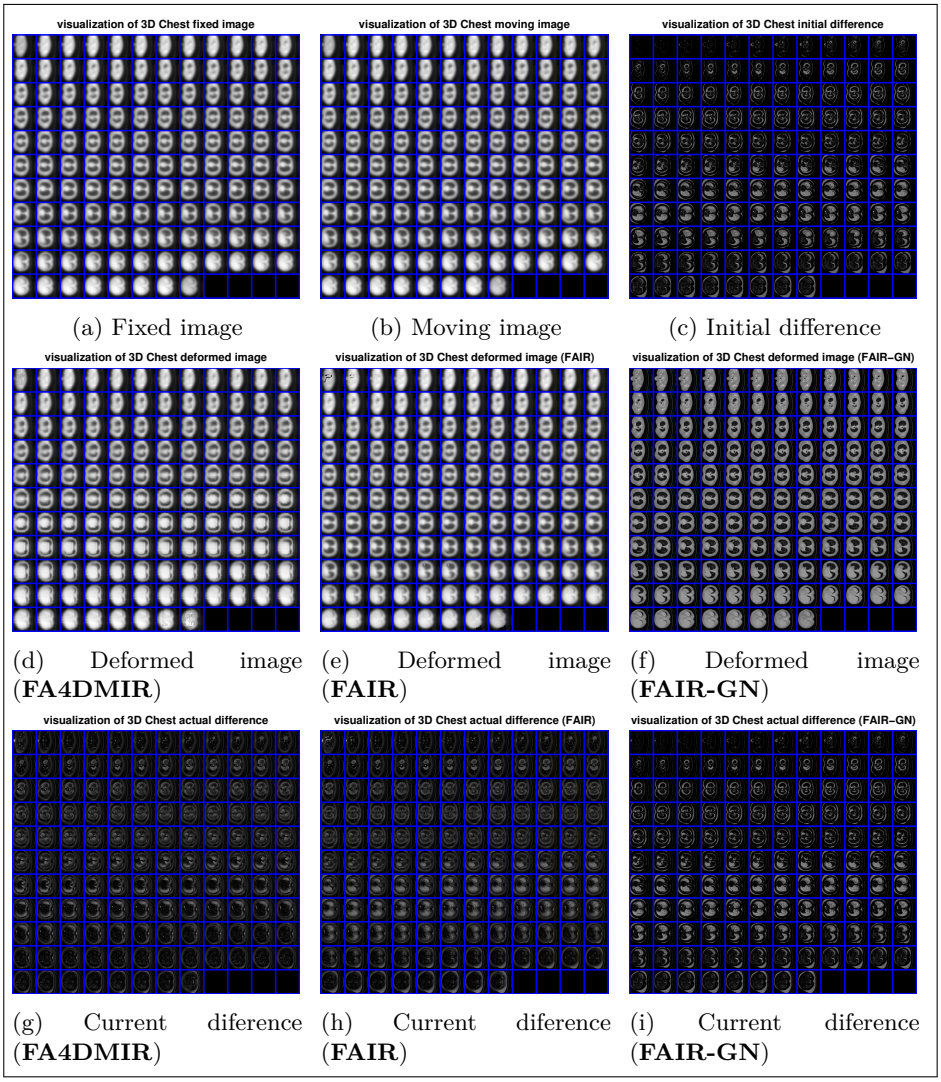
	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$8.2\,10^{4}$	1	0
FAIR	$2.3\,10^{3}$	$2.5\,10^{-2}$	$5.1\,10^{2}$
FAIR-GN	$1.0\,10^{-1}$	$1.0\,10^{-6}$	$4.3\,10^{3}$
FA4DMIR	$2.1\,10^{3}$	$2.3\,10^{-2}$	$5.1\,10^{3}$
Level 5			
From	$9.1\,10^{5}$	1	0
FAIR	$2.1\,10^{4}$	$2.4\,10^{-2}$	$1.2\,10^{3}$
FAIR-GN	$8.9\,10^{-1}$	$9.7\,10^{-7}$	$2.1\,10^{4}$
FA4DMIR	$2.2\,10^{4}$	$2.3\,10^{-2}$	$3.2\,10^{3}$
Level 6			
From	$1.1\,10^{7}$	1	0
FAIR	$7.1\,10^{5}$	$6.4\,10^{-2}$	$8.9\,10^{4}$
FAIR-GN	$5.0\,10^{5}$	$4.5\,10^{-2}$	$1.3\,10^{5}$
FA4DMIR	$8.1\,10^{4}$	$7.3\,10^{-3}$	$8.2\,10^{4}$
Level 7			
From	$6.2\,10^{7}$	1	0
FAIR	$7.2\,10^{6}$	$1.1\,10^{-1}$	$1.4\,10^{5}$
FAIR-GN	$4.5\,10^{6}$	$7.2\,10^{-2}$	$3.1\,10^{5}$
FA4DMIR	$1.3\,10^{6}$	$2.0\,10^{-2}$	$1.1\,10^{5}$

Table 6.12: Chest images: for each level of the chest images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.

Figure 6.24 presents a sample of results for the chest image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

6.3.7 The neurocranium (braincase) image

The $3D$ neurocranium images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size



(a) Fixed image (b) Moving image (c) Initial difference

(d) Deformed image (**FA4DMIR**) (e) Deformed image (**FAIR**) (f) Deformed image (**FAIR-GN**)

(g) Current diference (**FA4DMIR**) (h) Current diference (**FAIR**) (i) Current diference (**FAIR-GN**)

Figure 6.24: Chest image registration results level 7

[256, 256, 64] and they are registered on four levels as presented in the Table 6.13.

In this section we analyze the registration on 4 levels: $4, 5, 6$ and 7 in the aim of comparison with other images. Figure 6.25 presents the decrease of the function value with respect to the number of iterations for each of the three algorithms. The FAIR-GN algorithm perform better that the other at any level. Contrary to expectation, the FA4DMIR algorithm is worse algorithm at any level. However, the gap to the best algorithm, here the FAIR-GN decreases with the increase of the problem size.

Figure 6.26 presents the cumulated computation time after 100 iterations on the

Levels	Size	System size	Problem id
4	$(16, 16, 4)$	3 072	NeurocraniumLev4
5	$(32, 32, 8)$	24 576	NeurocraniumLev5
6	$(64, 64, 16)$	196 608	NeurocraniumLev6
7	$(128, 128, 32)$	1 572 864	NeurocraniumLev7
8	$(256, 256, 64)$	12 582 912	NeurocraniumLev8

Table 6.13: Neurocranium images: five levels of a couple of neurocranium images in 3D. These images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department.

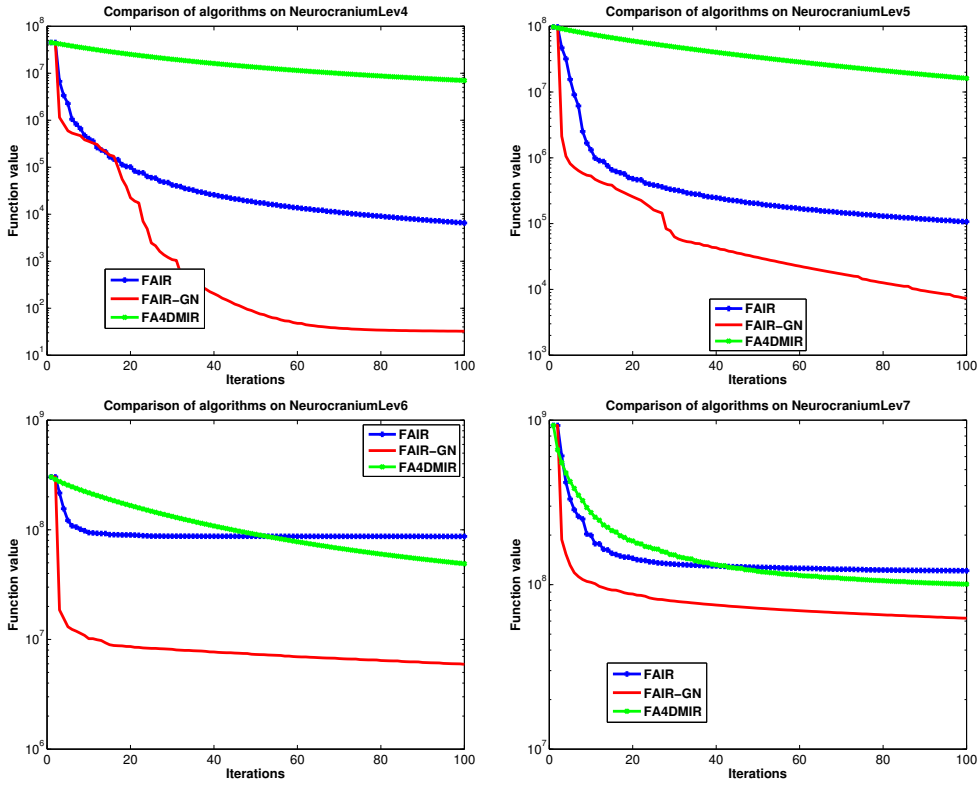


Figure 6.25: Neurocranium images. Comparison of the decrease of the function value with respect to the number of iterations on NeuroCranium images at fixed levels: level 4 (top plots) and level 5 (bottom plots).

NeuroCranium images. One may observe that the FA4DMIR algorithm that is the most time consuming at level 4 becomes more and more faster as the problem size is enlarged. It becomes the faster level 7 (bottom right).

Table 6.14 summarizes the results reached by the algorithms presented in Fig-

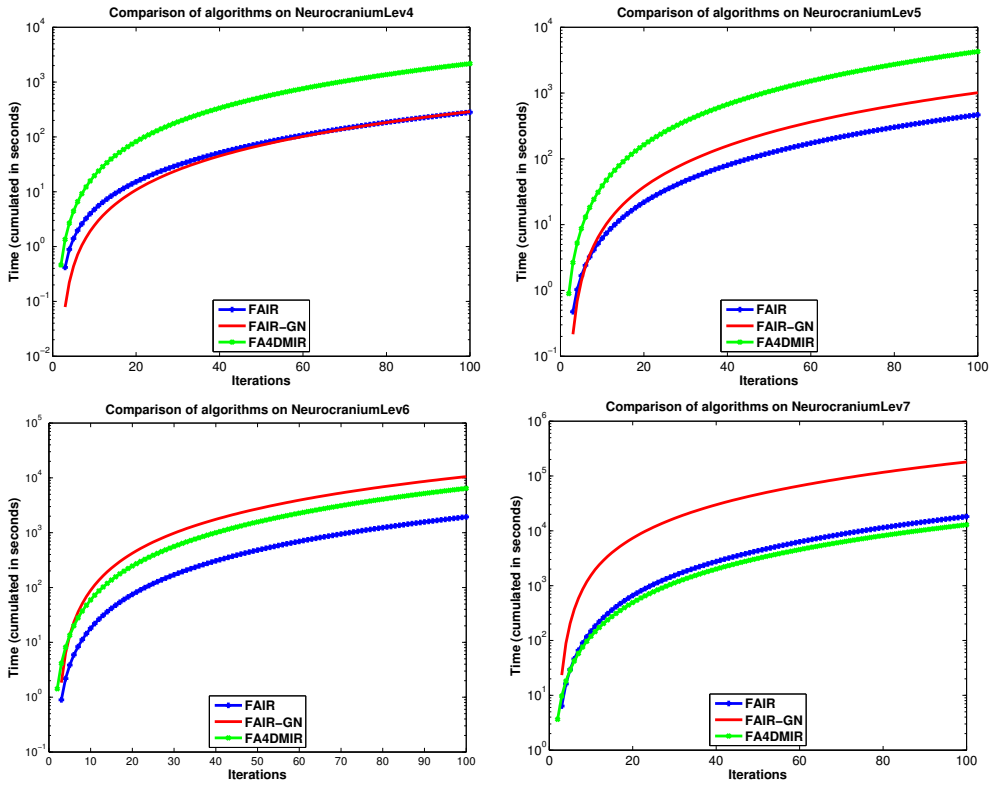


Figure 6.26: Comparison of the decrease of the function value with respect to the number of iterations on NeuroCranium images at fixed levels: level 6 (top plots) and level 7 (bottom plots).

ure 6.25 and Figure 6.26 . Runing 100 iterations for each level on the Neurocranium images images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c).

Figure 6.27 presents a sample of results for the neurcranium image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$7.2\,10^{7}$	1	0
FAIR	$1.2\,10^{4}$	$1.6\,10^{-4}$	$1.2\,10^{2}$
FAIR-GN	$4.0\,10^{1}$	$5.6\,10^{-7}$	$1.2\,10^{2}$
FA4DMIR	$4.1\,10^{7}$	$5.6\,10^{-1}$	$1.3\,10^{3}$
Level 5			
From	$1.0\ 10^{8}$	1	0
FAIR	$1.2\,10^{5}$	$1.2\,10^{-3}$	$2.2\,10^{2}$
FAIR-GN	$8.9\,10^{3}$	$8.9\,10^{-5}$	$8.1\,10^{2}$
FA4DMIR	$4.2\,10^{7}$	$4.2\,10^{-1}$	$5.2\,10^{3}$
Level 6			
From	$5.1\ 10^{8}$	1	0
FAIR	$1.2\,10^{8}$	$2.3\,10^{-1}$	$8.9\,10^{2}$
FAIR-GN	$1.3\,10^{7}$	$2.5\,10^{-2}$	$6.8\,10^{3}$
FA4DMIR	$8.3\,10^{7}$	$1.6\,10^{-1}$	$4.5\,10^{3}$
Level 7			
From	$1.0\ 10^{9}$	1	0
FAIR	$1.8\,10^{8}$	$1.8\,10^{-1}$	$8.9\,10^{4}$
FAIR-GN	$8.3\,10^{7}$	$8.3\,10^{-2}$	$1.8\,10^{5}$
FA4DMIR	$1.1\,10^{8}$	$1.1\,10^{-1}$	$8.4\,10^{5}$

Table 6.14: Neurocranium images: for each level of the neurocranium images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.

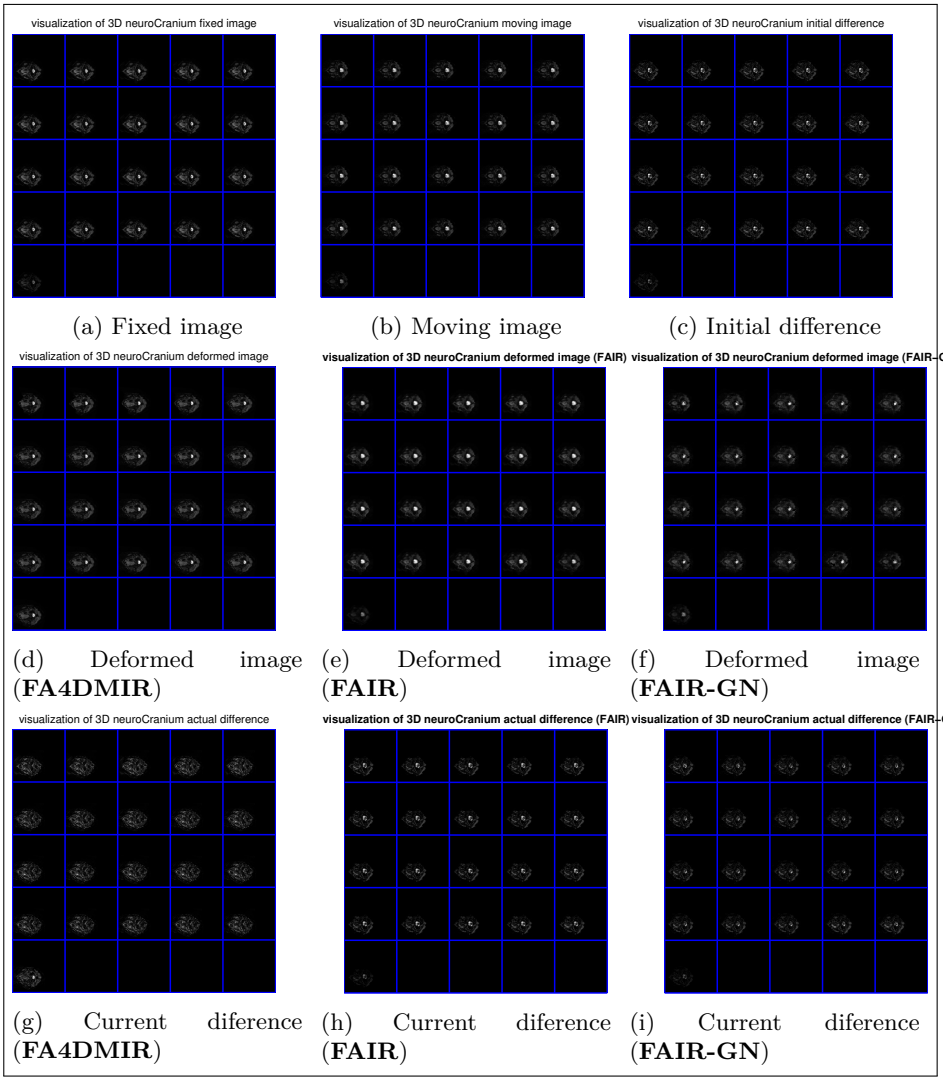
6.3.8 The lung image

The $3D$ lung images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size [512, 512, 1024] and they are registered on six levels as presented in the Table 6.15.

Levels	Size	System size	Problem id
4	$(16, 16, 32)$	24 576	LungLev4
5	$(32, 32, 64)$	196 608	lungLev5
6	$(64, 64, 128)$	1 572 864	LungLev6
7	$(128, 128, 256)$	12 582 912	LungLev7
8	$(256, 256, 512)$	100 663 296	LungLev8
9	$(512, 512, 1024)$	805 306 368	LungLev9

Table 6.15: Lung images. Six levels of a couple of lung images in 3D. These images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department.

Figure 6.28 presents the decrease of the functional value with respect to the num-



(a) Fixed image (b) Moving image (c) Initial difference

(d) Deformed image (**FA4DMIR**) (e) Deformed image (**FAIR**) (f) Deformed image (**FAIR-GN**)

(g) Current diference (**FA4DMIR**) (h) Current diference (**FAIR**) (i) Current diference (**FAIR-GN**)

Figure 6.27: Neurocranium image registration results level 7

ber of iterations for registration of lung images. It is observed that the FAIR-GN algorithm (red line) is to be preferred for these images since it performs well than the others. However, the FA4DMIR algorithm shows that the larger is the problem size, the more it is expected to perform better. This is shown at level 7 (bottom right) it becomes better.

Figure **??** illustrates the cumulated computation time after 100 iterations for lung image registration. It is observed further that FA4DMIR algorithm (green line) is time consuming at low levels but becomes the faster than FAIR and FAI-GN at high

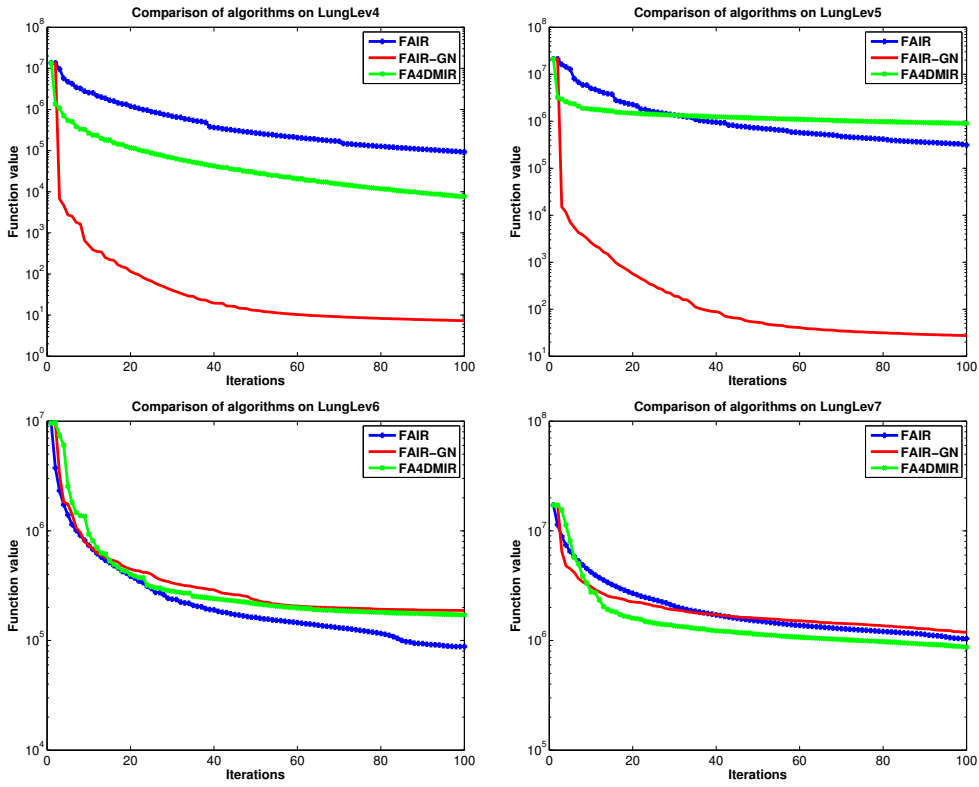


Figure 6.28: Lung images: comparison of the function value decrease with respect to the number of iterations on lung images at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

level.

Table 6.16 summarizes the results reached by the algorithms presented in Figure 6.28 and Figure 6.29 .

Figure 6.30 presents a sample of results for the lung image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

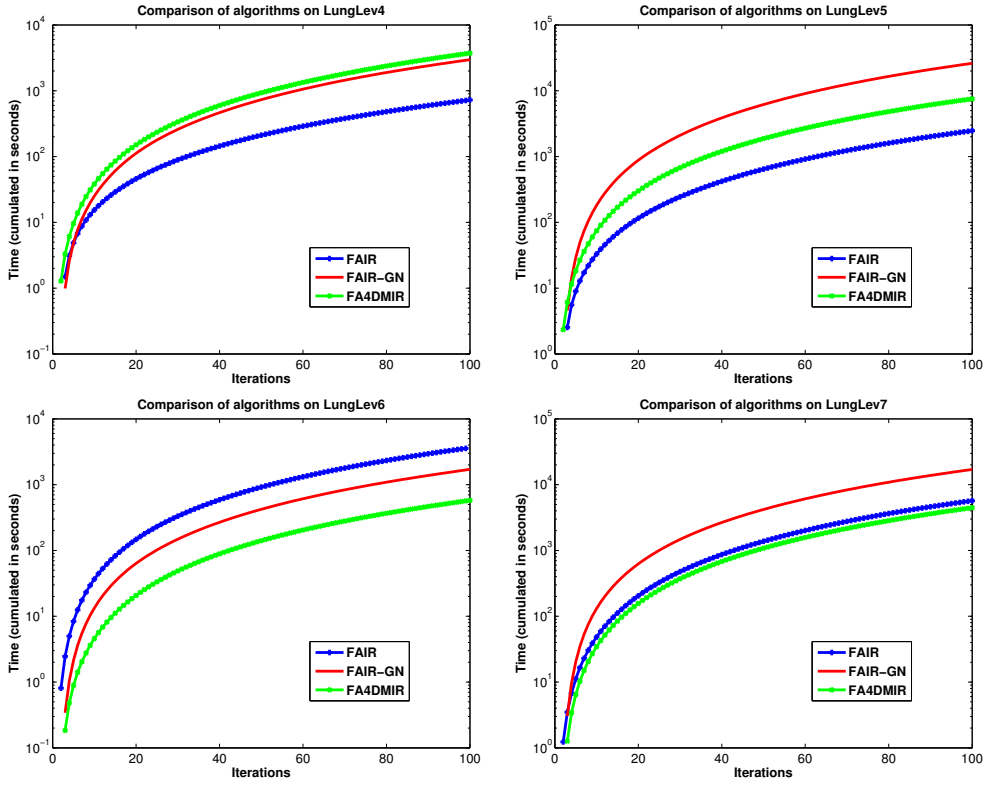
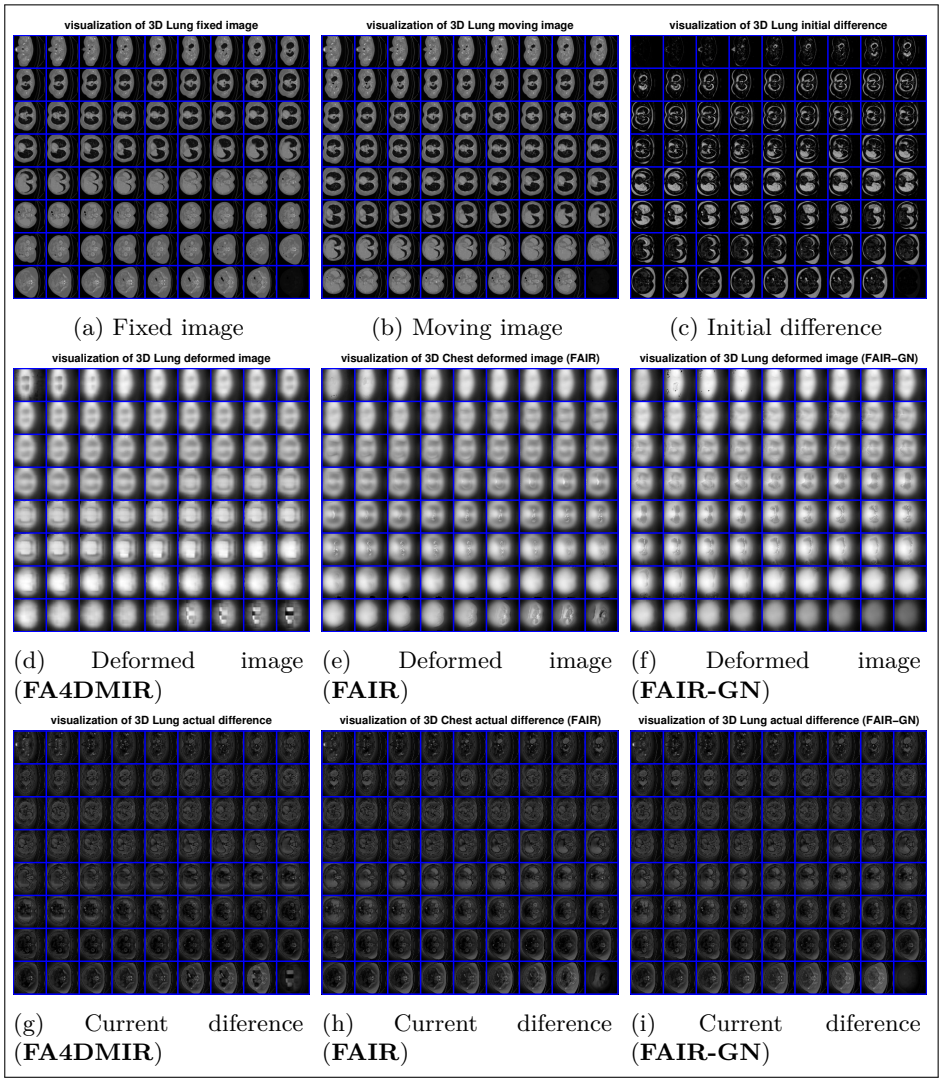


Figure 6.29: Lung images: comparison of the cumulated computation time with respect to the number of iterations on lung images at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$1.5\,10^{7}$	1	0
FAIR	$1.2\,10^{5}$	$8\,10^{-3}$	$2.2\,10^{2}$
FAIR-GN	$9.5\,10^{0}$	$6.3\,10^{-7}$	$3.5\,10^{3}$
FA4DMIR	$9.8\,10^{3}$	$6.5\,10^{-4}$	$4.6\,10^{3}$
Level 5			
From	$4.1\,10^{7}$	1	0
FAIR	$1.0\,10^{6}$	$2.4\,10^{-2}$	$1.7\,10^{3}$
FAIR-GN	$3.5\,10^{1}$	$8.5\,10^{-7}$	$1.3\,10^{4}$
FA4DMIR	$5.0\,10^{6}$	$1.2\,10^{-1}$	$5.2\,10^{3}$
Level 6			
From	$1.0\,10^{7}$	1	0
FAIR	$9.7\,10^{4}$	$9.7\,10^{-3}$	$2.5\,10^{4}$
FAIR-GN	$3.1\,10^{5}$	$3.1\,10^{-2}$	$1.7\,10^{4}$
FA4DMIR	$3.0\,10^{5}$	$3.0\,10^{-2}$	$1.1\,10^{4}$
Level 7			
From	$2.3\,10^{7}$	1	0
FAIR	$1.4\,10^{6}$	$6.0\,10^{-2}$	$4.1\,10^{4}$
FAIR-GN	$1.6\,10^{6}$	$6.9\,10^{-2}$	$4.1\,10^{5}$
FA4DMIR	$1.0\,10^{6}$	$4.3\,10^{-2}$	$4.0\,10^{4}$

Table 6.16: Lung images: for each level, this table presents the reached function value (F_c), the ratio F_c/F_0 where F_c is the current function value and F_0 the initial function value and the cumulated time time (T_c) for 100 iterations.



(a) Fixed image (b) Moving image (c) Initial difference

(d) Deformed image (**FA4DMIR**) (e) Deformed image (**FAIR**) (f) Deformed image (**FAIR-GN**)

(g) Current diference (**FA4DMIR**) (h) Current diference (**FAIR**) (i) Current diference (**FAIR-GN**)

Figure 6.30: Lung image registration results level 7

6.3.9 The phantom image

The $3D$ phantom images here are available in the FAIR package. The images are of size [128, 64, 128] and were given on the domain $\Omega = [0, 64, 0, 64, 0, 64]$. They are registered on four levels as presented in the Table **??**.

Levels	Size	System size	Problem id
4	$(16, 8, 16)$	6 144	PhantomLev4
5	$(32, 16, 32)$	49 152	PhantomLev5
6	$(64, 32, 64)$	393 216	PhantomLev6
7	$(128, 64, 128)$	3 145 728	PhantomLev7

Table 6.17: Phantom images. Four levels of a couple of phantom images in 3D. These images are available in the FAIR package.

Figure 6.31 presents the decrease of the functional value with respect to the number of iterations on the phantom images registration. It is observed that the FA4DMIR algorithm (green curve) performs better than FAIR algorithm (blue curve) and than FAIR-GN algorithms at any level. Observe that this is true even for relativelly small images: level 4 (top left) and level 5 (top right). In fact, these phantom images, in addition their favourable nature (CT-scan) were very sparse.

Figure 6.32 visualizes the cumulated computation time after 100 iterations for lung images registration. One can observe that, although the images are sparse, the FA4DMIR algorithm (green line) remains less competitive at low levels: level 4 (top left) and level 5 (top right) while it becomes competitive for relatively high levels: level 6 (bottom left) and level 7 (bottom right).

Table 6.18 prensents the summary of reached values by these three algorithms on phantom images. This table presents summarizes results from Figure 6.31 and Figure 6.32.

Figure 6.33 presents a sample of results for the neurcranium image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels $(4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

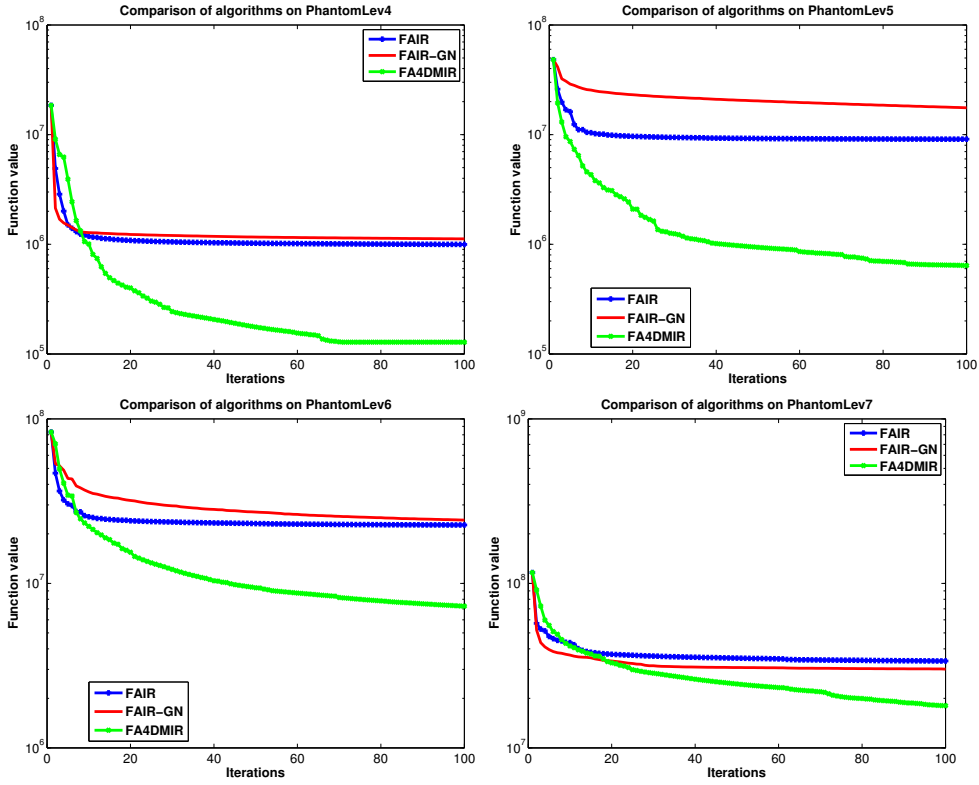


Figure 6.31: Phantom images: comparison of the functional value decrease with respect to the number of iterations on phantom images at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

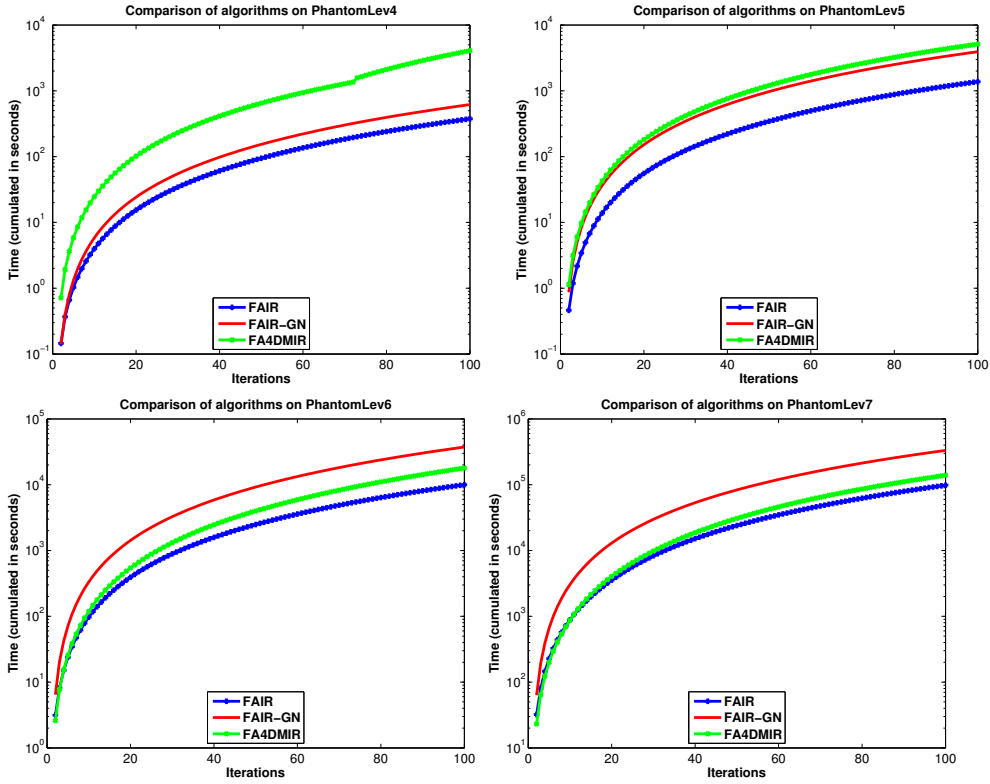
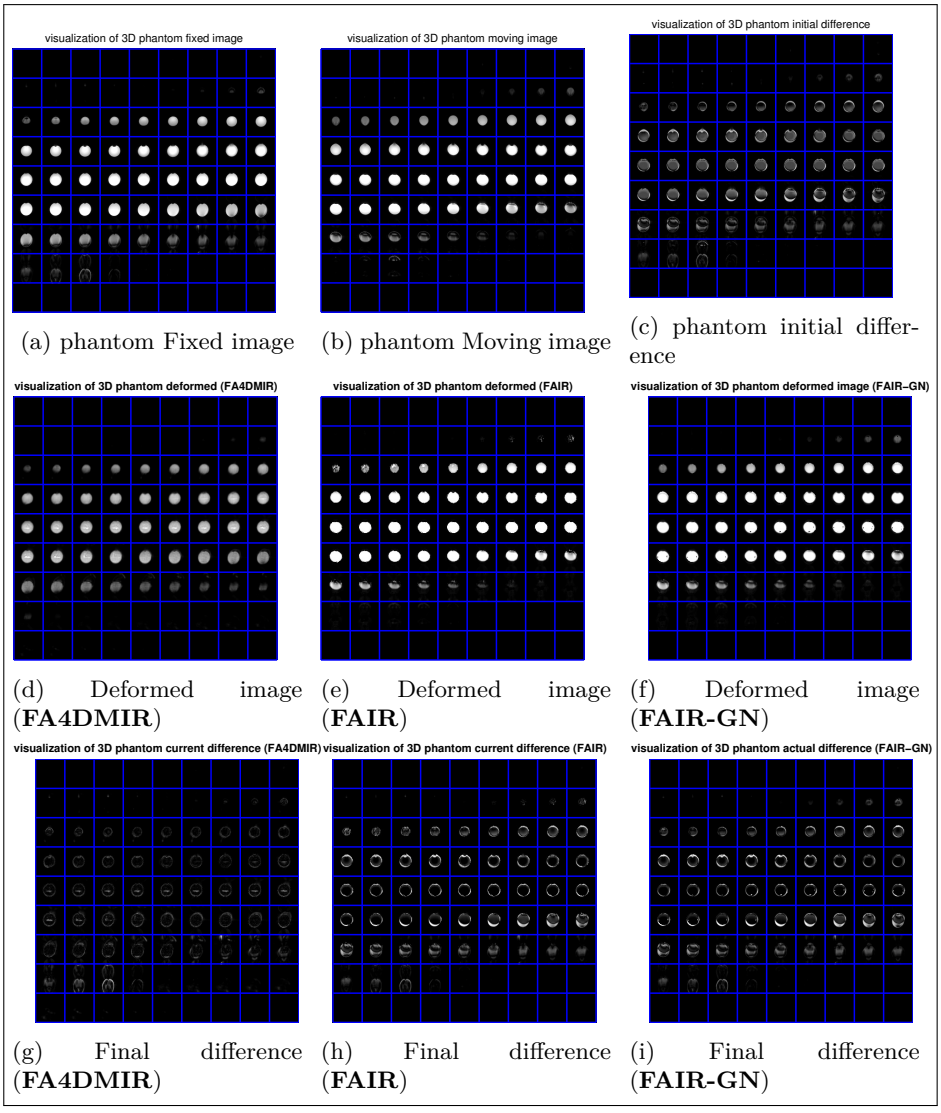


Figure 6.32: Phantom images: comparison of the cumulated computation time with respect to the number of iterations on phantom images at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$4.8\,10^{7}$	1	0
FAIR	$9.0\,10^{6}$	$1.8\,10^{-1}$	$1.0\,10^{2}$
FAIR-GN	$1.7\,10^{7}$	$3.5\,10^{-1}$	$1.4\,10^{2}$
FA4DMIR	$1.4\,10^{5}$	$2.9\,10^{-3}$	$4.6\,10^{3}$
Level 5			
From	$7.2\,10^{7}$	1	0
FAIR	$2.2\,10^{7}$	$3.0\,10^{-1}$	$8.0\,10^{2}$
FAIR-GN	$4.2\,10^{7}$	$5.8\,10^{-1}$	$5.8\,10^{3}$
FA4DMIR	$6.7\,10^{5}$	$9.3\,10^{-3}$	$6.0\,10^{3}$
Level 6			
From	$8.3\,10^{7}$	1	0
FAIR	$2.3\,10^{7}$	$2.7\,10^{-1}$	$5.3\,10^{3}$
FAIR-GN	$2.4\,10^{7}$	$2.8\,10^{-1}$	$4.3\,10^{4}$
FA4DMIR	$1.2\,10^{7}$	$1.4\,10^{-1}$	$7.8\,10^{3}$
Level 7			
From	$1.1\,10^{8}$	1	0
FAIR	$6.3\,10^{7}$	$5.7\,10^{-1}$	$9.6\,10^{4}$
FAIR-GN	$6.0\,10^{7}$	$5.4\,10^{-1}$	$5.0\,10^{5}$
FA4DMIR	$2.7\,10^{7}$	$2.4\,10^{-1}$	$9.2\,10^{4}$

Table 6.18: Phantom images: for each level of phantom images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.



(a) phantom Fixed image (b) phantom Moving image (c) phantom initial difference

(d) Deformed image (**FA4DMIR**) (e) Deformed image (**FAIR**) (f) Deformed image (**FAIR-GN**)

(g) Final difference (**FA4DMIR**) (h) Final difference (**FAIR**) (i) Final difference (**FAIR-GN**)

Figure 6.33: Phantom image registration results (sample)

6.3.10 The PET-CT1 image

The $3D$ PET-CT1 images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size [256, 256, 64] and they are registered on five levels as presented in the Table 6.19.

Levels	Size	System size	Problem id
4	$(16, 16, 4)$	3 072	PET-CTLev4
5	$(32, 32, 8)$	24 576	PET-CTLev5
6	$(64, 64, 16)$	196 608	PET-CTLev6
7	$(128, 128, 32)$	1 572 864	PET-CTLev7
8	$(256, 256, 64)$	12 582 912	PET-CTLev8

Table 6.19: PET-CT1 images: five levels of a couple of PET-CT1 images images in 3D. These images were provided by Hubert Meurisse,CHU-UCL-Namur mont Godine hospital, nuclear medecine department.

Figure 6.34 presents the decrease of the function value with respect to the number of iterations on the PET-CT1 images registration. For these images, the FAIR4DMIR algorithm (in red) performs better at all levels (level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right)) than the FAIR algorithm (blue line) and FAIR-GN algorithm (green line). This is explained by the nature of the image and these images were sparse. Comparison between FAIR and FA4DMIR shows that the FA4DMIR algorithm perform well than the FAIR algorithm even for low levels (small size images).

Figure 6.35 presents the cumulated computation time after 100 iterations on PET-CT1 images. It is observed that FA4DMIR (green line) is faster than FAIR-GN (red line) and FAIR (blue line) for all the levels (all the plots).

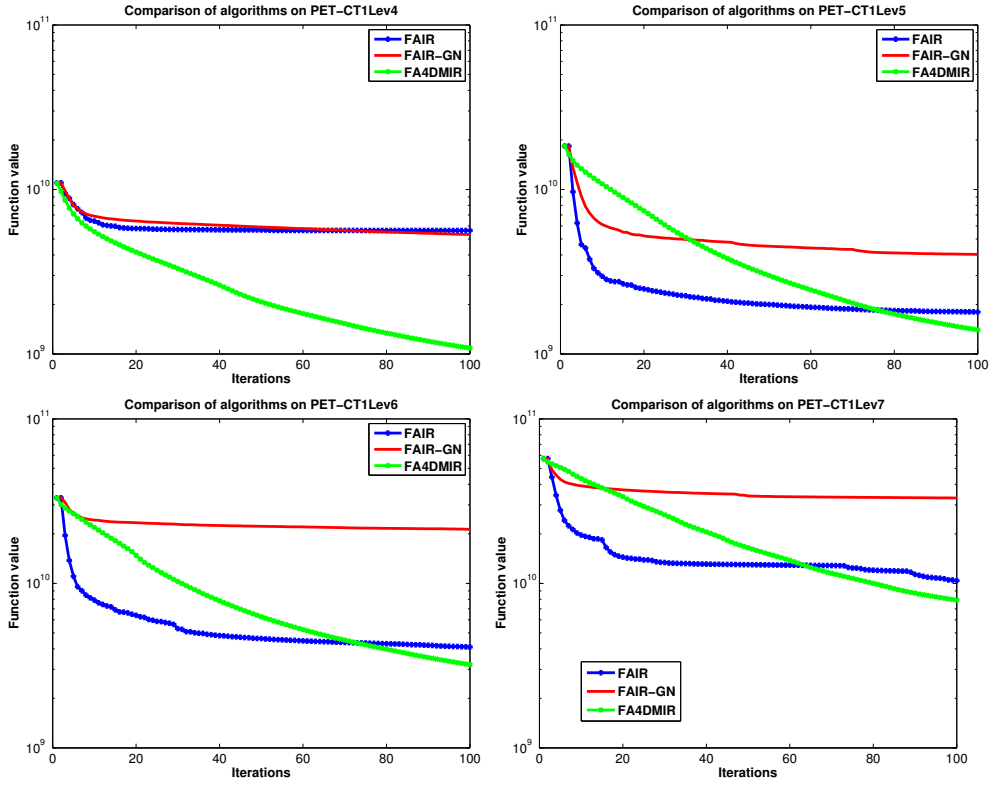


Figure 6.34: PET-CT1 images: comparison of the fuction value decrease with respect to the number of iterations at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

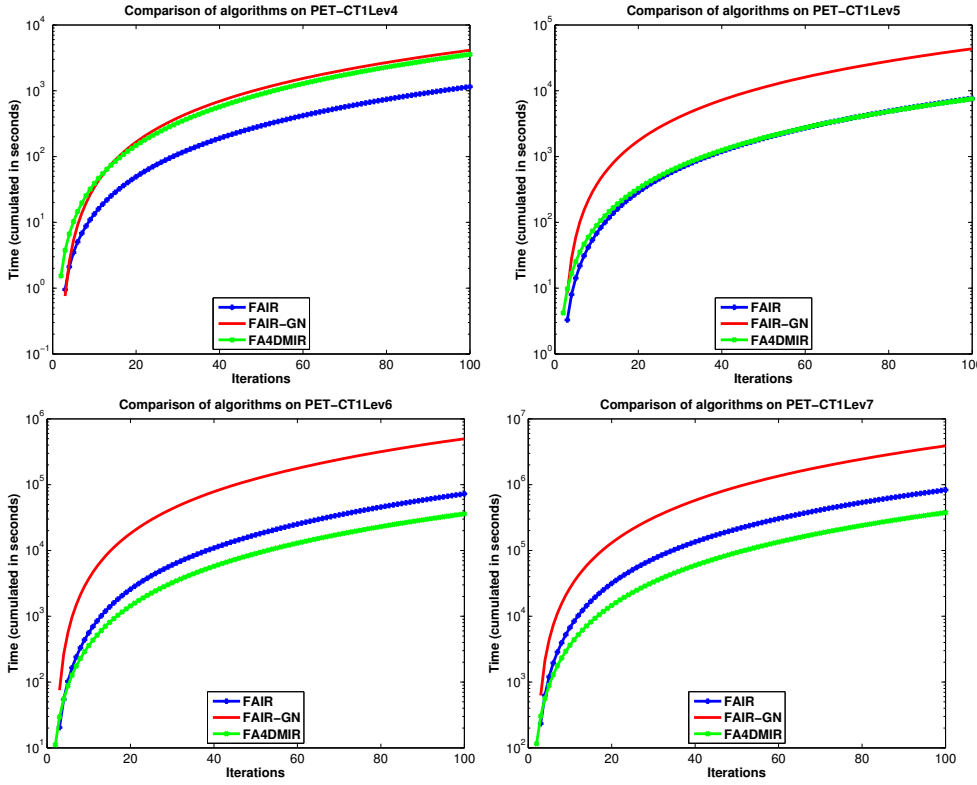


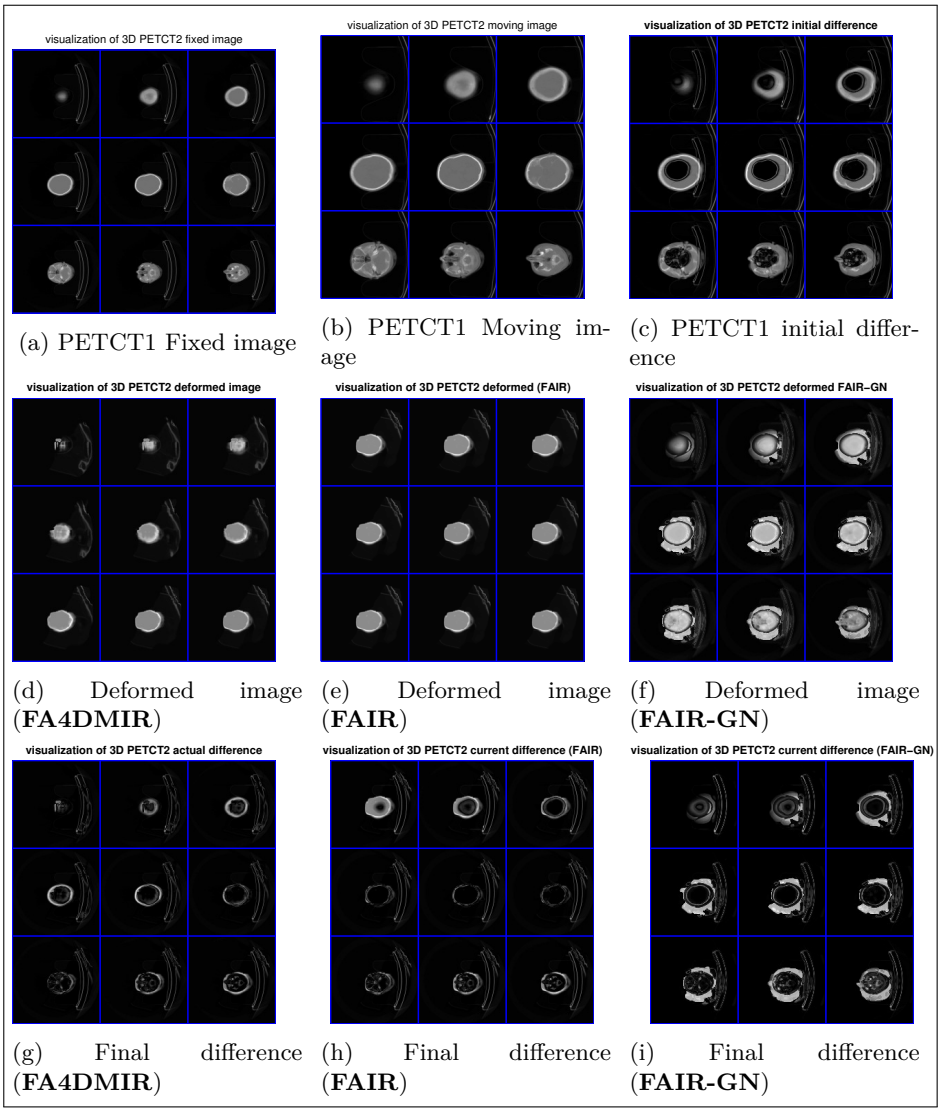
Figure 6.35: PET-CT1 images: comparison of the cumulated computation time with respect to the number of iterations at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

Table 6.20 prensents the summary of reached values by these three algorithms on phantom images. This table presents summarizes results from Figure 6.34 and Figure 6.35.

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$1.1\,10^{10}$	1	0
FAIR	$6.2\,10^{9}$	$5.6\,10^{-1}$	$5.2\,10^{2}$
FAIR-GN	$6.0\,10^{9}$	$5.4\,10^{-1}$	$5.3\,10^{3}$
FA4DMIR	$1.0\,10^{9}$	$9.0\,10^{-2}$	$5.2\,10^{3}$
Level 5			
From	$4.8\,10^{10}$	1	0
FAIR	$2.0\,10^{9}$	$4.1\,10^{-2}$	$1.1\,10^{3}$
FAIR-GN	$5.2\,10^{9}$	$1.0\,10^{-1}$	$3.1\,10^{4}$
FA4DMIR	$1.2\,10^{9}$	$2.5\,10^{-2}$	$1.1\,10^{3}$
Level 6			
From	$2.4\,10^{10}$	1	0
FAIR	$5.3\,10^{9}$	$2.2\,10^{-1}$	$5.3\,10^{3}$
FAIR-GN	$2.1\,10^{10}$	$8.7\,10^{-1}$	$4.3\,10^{4}$
FA4DMIR	$4.2\,10^{9}$	$1.7\,10^{-1}$	$9.1\,10^{3}$
Level 7			
From	$7.8\,10^{10}$	1	0
FAIR	$1.6\,10^{10}$	$2.0\,10^{-1}$	$1.1\,10^{5}$
FAIR-GN	$6.1\,10^{10}$	$7.8\,10^{-1}$	$3.1\,10^{6}$
FA4DMIR	$1.0\,10^{10}$	$1.2\,10^{-1}$	$8.2\,10^{4}$

Table 6.20: PET-CT1Images images: for each level of PET-CT1 images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.

Figure 6.36 presents a sample of results for the PET-CT1 image registration, within a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).



(a) PETCT1 Fixed image

(b) PETCT1 Moving image

(c) PETCT1 initial difference

(d) Deformed image (**FA4DMIR**)

(e) Deformed image (**FAIR**)

(f) Deformed image (**FAIR-GN**)

(g) Final difference (**FA4DMIR**)

(h) Final difference (**FAIR**)

(i) Final difference (**FAIR-GN**)

Figure 6.36: PET-CT1 image registration results

6.3.11 The PET-CT2 image

The 3D pet-ctBouge images were provided by Hubert Meurisse, CHU-UCL-Namur mont Godine hospital, nuclear medecine department. The images are of size [512, 512, 1024] and they are registered on six levels as presented in the Table **??**.

Levels	Size	System size	Problem id
4	$(16, 16, 32)$	24 576PET-CTBLev4	
5	$(32, 32, 64)$	196 608	PET-CTBLev5
6	$(64, 64, 128)$	1 572 864	PET-CTBLev6
7	$(128, 128, 256)$	12 582 912	PET-CTBLev7
8	$(256, 256, 1024)$	100 663 296	PET-CTBLev8
9	$(512, 512, 1024)$	805 306 368	PET-CTBLev9

Figure 6.37 presents the decrease of the functional value with respect to the number of iterations for registration of PET-CT2 images. It is observed that the FAIR algorithm (blue line) is to be preferred for these images since it performs well than the others at any level except the level 4 (top left) where the FAIR-GN performs better. For these images, the FA4DMIR algorithm is the worse, even for high levels. This may be explained by the fact that the images are sensitive to truncation and, in addition, they were very dense. Figure 6.38 visualizes the cumulated computation time after 100 iterations for PET-CT2 images registration. AS expected, one can observe that the FA4DMIR algorithm (green line) remains less competitive at low levels: level 4 (top left) and level 5 (top right) while it becomes competitive for relatively high levels: level 6 (bottom left) and level 7 (bottom right).

Table 6.21 prensents the summary of reached values by these three algorithms on PET-CT2 images. This table summarizes results from Figure 6.37 and Figure 6.38.

Figure 6.39 presents a sample of results for the PET-CT1 image registration, with a mulitilevel approach. The shown results concern the level 7. At low levels ($4, 5$ and 6) we run 100 iterations while at level 7 we run 10 iterations to save time. Top row: fixed image (left), moving image or image to be deformed (middle) and the initial difference (right). Middle row: the deformed image by the FA4DMIR algorithm (left), the deformed image by the FAIR algorithm (middle) and the deformed image by the FAIR-GN algorithm (right). Bottom row: the current difference after registration by FA4DMIR algorithm (left), the current difference after registration by FAIR algorithm (middle) and the current difference after registration FAIR-GN algorithm (right).

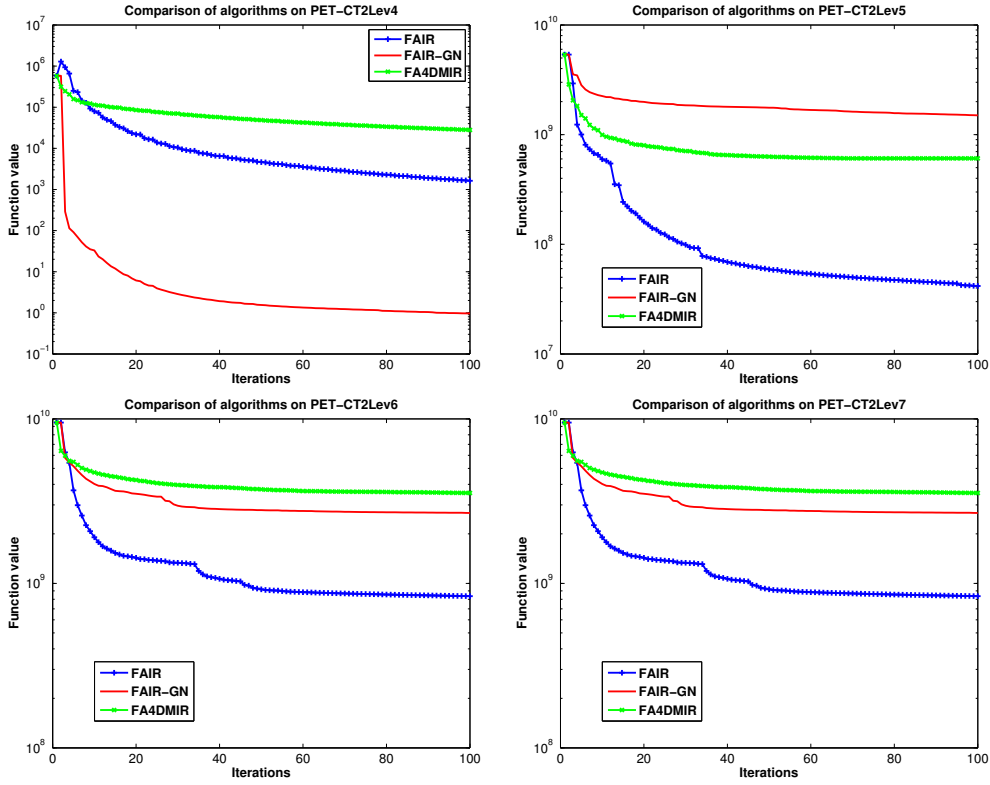


Figure 6.37: PET-CT2 image: comparison of the decrease of the function value with respect to the number of iterations at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

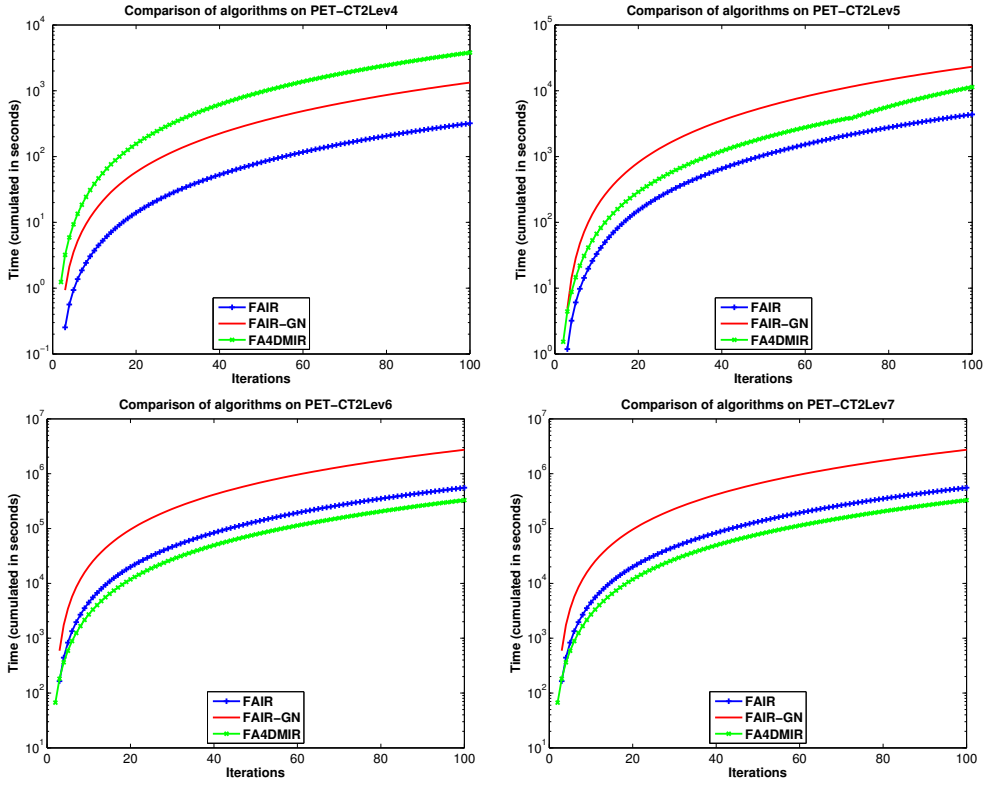
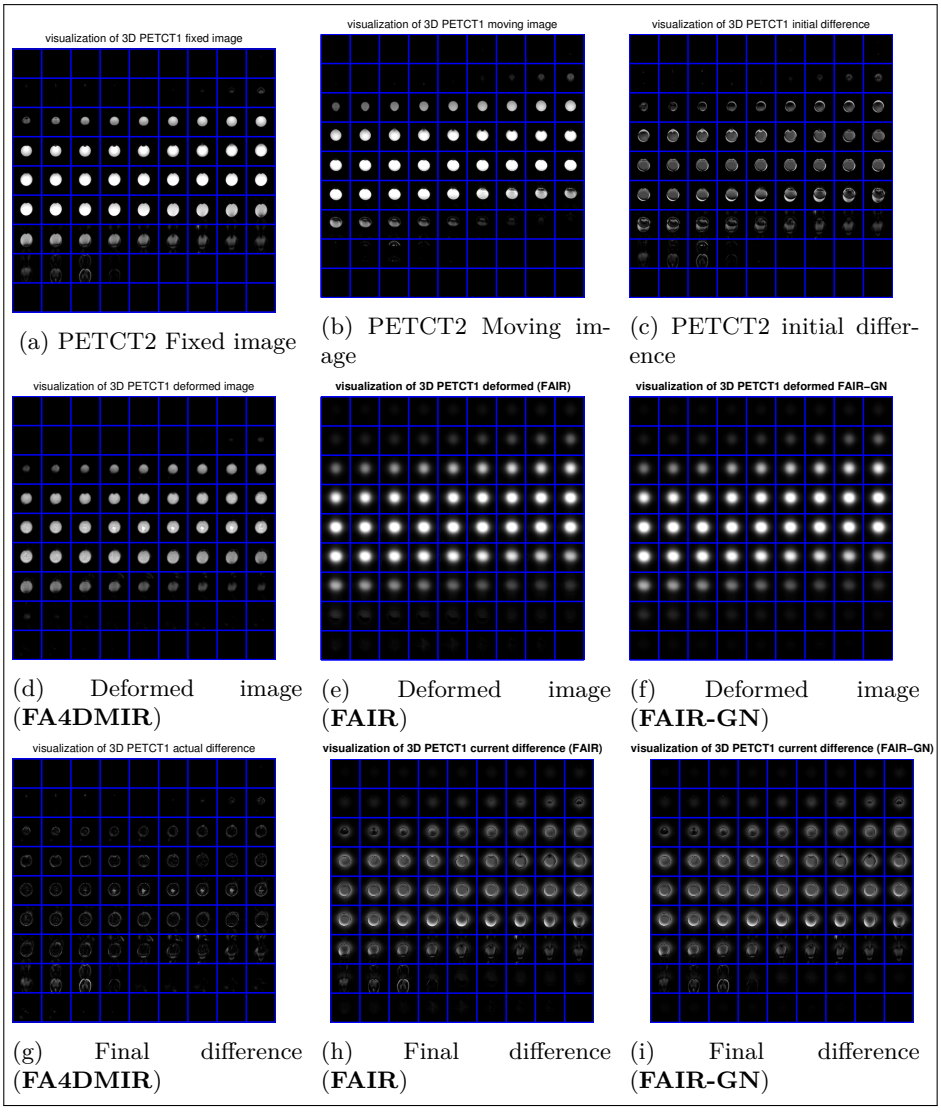


Figure 6.38: PET-CT2 image: comparison of the cumulated computation time with respect to the number of iterations at fixed levels: level 4 (top left), level 5 (top right), level 6 (bottom left) and level 7 (bottom right).

	$\mathbf{F_c}$	$\mathbf{F_c/F_0}$	$\mathbf{T_c}$
Level 4			
From	$1.0\,10^{6}$	1	0
FAIR	$1.0\,10^{4}$	$1.0\,10^{-2}$	$1.2\,10^{2}$
FAIR-GN	$1.2\,10^{0}$	$1.2\,10^{-6}$	$9.3\,10^{2}$
FA4DMIR	$2.0\,10^{5}$	$2.0\,10^{-1}$	$3.6\,10^{3}$
Level 5			
From	$8.8\,10^{9}$	1	0
FAIR	$7.8\,10^{7}$	$8.9\,10^{-3}$	$2.1\,10^{3}$
FAIR-GN	$2.5\,10^{9}$	$2.8\,10^{-1}$	$2.1\,10^{4}$
FA4DMIR	$1.0\,10^{9}$	$1.1\,10^{-1}$	$7.1\,10^{3}$
Level 6			
From	$1.0\,10^{10}$	1	0
FAIR	$8.1\,10^{8}$	$8.1\,10^{-1}$	$1.0\,10^{5}$
FAIR-GN	$5.9\,10^{9}$	$5.0\,10^{-1}$	$8.5\,10^{5}$
FA4DMIR	$6.1\,10^{9}$	$6.1\,10^{-1}$	$8.4\,10^{4}$
Level 7			
From	$1.0\,10^{10}$	1	0
FAIR	$9.8\,10^{8}$	$9.8\,10^{-2}$	$1.1\,10^{5}$
FAIR-GN	$5.2\,10^{9}$	$5.2\,10^{-1}$	$1.1\,10^{6}$
FA4DMIR	$7.2\,10^{9}$	$7.2\,10^{-1}$	$9.2\,10^{4}$

Table 6.21: PETCT2 images: for each level of PET-CT2 images, this table presents the reached function value (F_c), the ratio F_c/F_0 and the cumulated time (T_c) for 100 iterations.



(a) PETCT2 Fixed image

(b) PETCT2 Moving image

(c) PETCT2 initial difference

(d) Deformed image (**FA4DMIR**)

(e) Deformed image (**FAIR**)

(f) Deformed image (**FAIR-GN**)

(g) Final difference (**FA4DMIR**)

(h) Final difference (**FAIR**)

(i) Final difference (**FAIR-GN**)

Figure 6.39: PETCT2 image registration results

6.3.12 Conclusion

From the observations above, one can conclude that for larger images within limited budget in term of computing time, the FA4DMIR algorithm is to be preferred. In opposition, for smaller images the FAIR-GN algorithm is advised, in the hope that the second order information will be captured by certain use of the Jacobian. Since the acceleration may not be guaranteed by this last, the FAIR algorithm is also usefull.

We have observed that, in the linear systems, the more the right-hand side is sparse, the more Tensor-Train methods are competitive. In addition, we argue that algorithms using Tensor-Train methods perform well for UltraSound (US), Computed Tomography (CT) and Positron Emission Tomography (PET) images, but are less good on Magnetic Resonance Images (MRI). This has to be verified by futher research using more test images.

Conclusion and perspectives

In this work, we have addressed optimization algorithms used to solve medical image registration problems. The focus was on the non-rigid and nonparametric registration problem for high-resolution 3D medical images. Based on the observation that this problem is highly time and memory consuming, we focused on studying algorithms and proposing efficient techniques in order to accelerate the registration process.

In the first chapter, we have proposed a large overview of image registration problem algorithms. Then we have focused on developing techniques to accelerate the image registration process. For this purpose, and based on the Flexible Algorithms for Image Registration package (FAIR), we have pointed out that the most expensive step in certain registration algorithms is the solution of linear systems arising in this process. Therefore we have proposed in chapter 4 certain preconditioners that may accelerate the resolution of the linear systems. We have found that, contrary to what one could expect, the Tchebychev polynomial preconditioner is among the most efficient preconditioners that may address the large linear systems from high dimensional image registration.

Since linear systems from image registration problems arise from PDEs discretization, they induce large, sparse and structured matrices, but in general ill-conditioned. Hence, we have developed in Chapter 5 how these large matrices can be addressed in numerical tensor representations and how the corresponding linear systems can be solved by efficient new techniques from numerical tensor computing. We have shown via numerical experiments in Chapter 6 how tensor methods may provide better compromise between speed and precision.

The final contribution of this thesis is an extension of the FAIR package by providing to the user, additional possibilities of speeding up the process. This includes the possibility to use a Tchebychev polynomial preconditioner in the PCG solver or to use a PCG solver in TT-Tensor format via a new algorithm that we propose to call Flexible Algorithm for Deformable Medical Image Registration (FA4DMIR (see 5.4)). In Chapter 6, we give some numerical illustrations on a set of real 3D image registration problems of the usefulness of the proposed approach and in which circumstances.

From the numerical experiments presented in Chapter 6 and from the literature, we may conclude that in the linear systems, the more the right-hand vector is

sparse, the more Tensor-Train methods are competitive. In addition, The Tensor-Train methods are well suited for parallel programming environment, better than in matrix methods. Thus, the FA4DMIR algorithm performs well for large 3D images with sparse structure in a parallel computing environment.

However, the Tensor-Train methods we have experimented still related on the fixed maximal rank. A small rank may speed up the process but may lead to non accurate results, while a high maximal rank may lead to very expensive algorithms. A compromise has to be done and we recommend to get insight on l-rank from tensorlab [50] to have an idea on the compressibility of the data.

The goal of the work was to design and develop efficient algorithms for 3D medical image registration. Although we have proposed interesting suggestions, we have found that registration algorithms for high 3D images is a challenging field where there remains many issues. Among our perspectives, we would like first to address an integrated way of using numerical tensor methods in the whole registration process. This may allow more speed since the transfer cost from one format to another is saved. Second, for polynomial preconditioners, we are interested in combining more preconditioners to better address both the structure of the matrix and the right-hand side of the system. In addition to this, we would like to work more closely with professionals in hospital and clinical applications to address problems with direct impact on the day-to-day life.

List of Tables

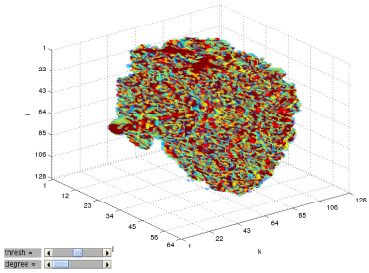
List of Figures

Appendices

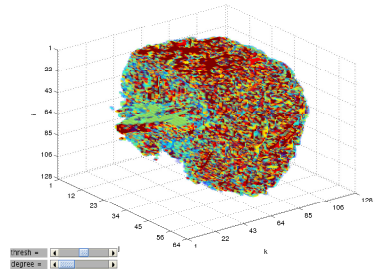
Appendix A

Visualization of images

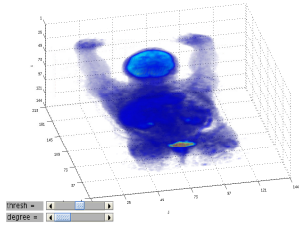
Here we visualize the images using the matlab command voxel3 from tensorlab. Fixed images are visualized at the left side and moving images are visualized at the right-hand side. We apologize for this poor visualization. More improvement for visualization is planned for future work.



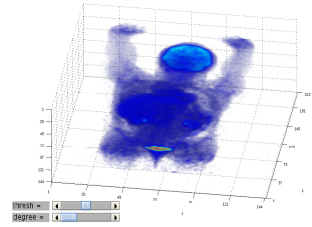
(a) Brain fixed image



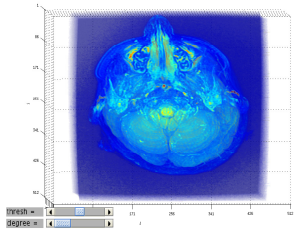
(b) Brain moving image



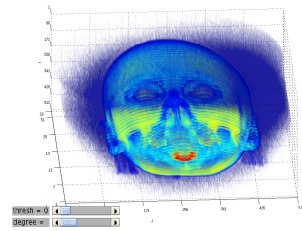
(c) Chest fixed image



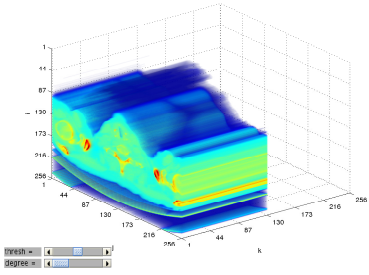
(d) Chest moving image



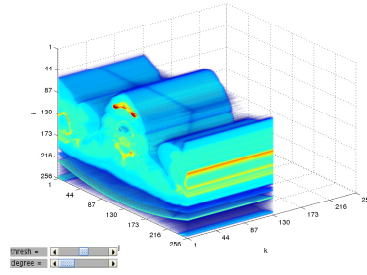
(e) Crane fixed image



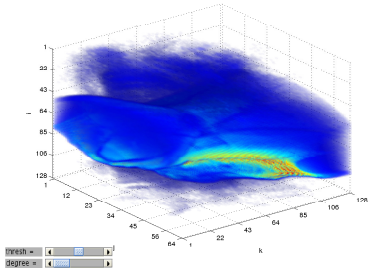
(f) Crane moving image



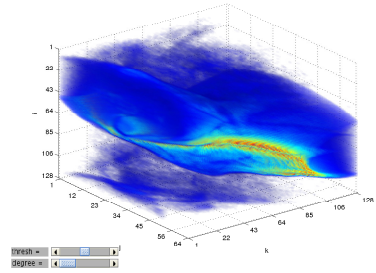
(g) Foetus fixed image



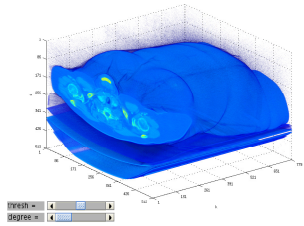
(h) Foetus moving image



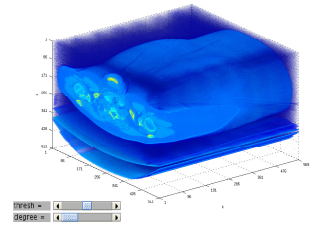
(i) Knee fixed image



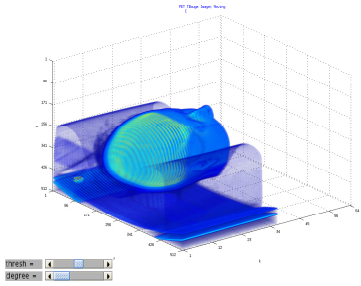
(j) Knee moving image



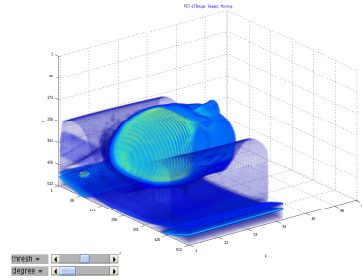
(k) Lung fixed image



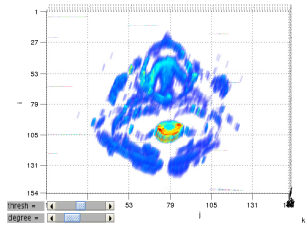
(l) Lung moving image



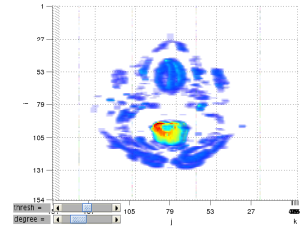
(m) neuroCranium fixed



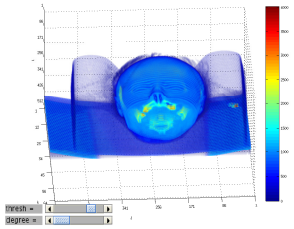
(n) neuroCranium moving



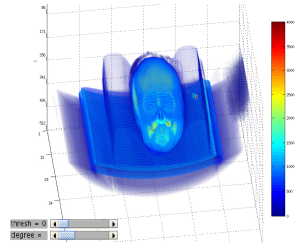
(o) PET-CT1 fixed image



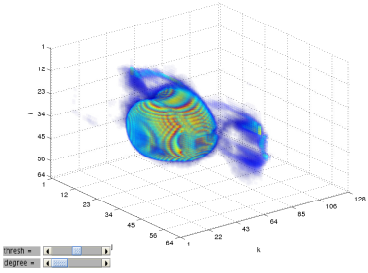
(p) PET-CT1 moving image



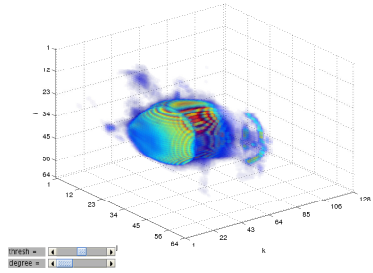
(q) PET-CT2 fixed image



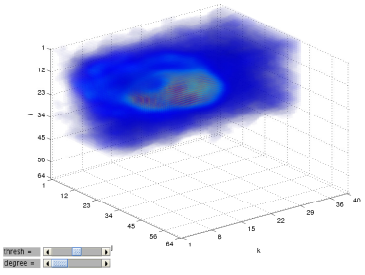
(r) PET-CT2 moving image



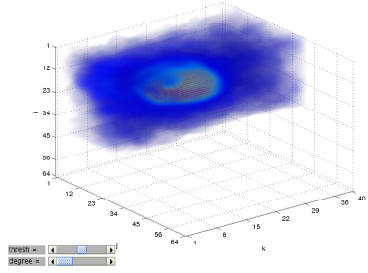
(s) Phantom fixed image



(t) Phantom moving image



(u) Mice fixed image



(v) Mice moving image

Appendix B

Discrete differential operators

Let us consider an image of size $[n^{(1)}, n^{(2)}, n^{(3)}]$ on a three dimensional domain ω. By simplicity let us assume $n^{(1)} = n^{(2)} = n^{(3)} = n$ and $\Omega = [0,1]^3$. For a given displacement field $u = \left[u^{(1)}, u^{(2)}, u^{(3)}\right]$, we may approximate its first derivatives following the backward finite differences methods. For this, we can apply the unidimensional first derivatives operator

$$\partial_n^h = \frac{1}{h}\begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{n\times n+1}. \tag{B.1}$$

where $h = h^{(1)} = h^{(2)} = h^{(3)} = \dfrac{1}{n}$ to each component of the displacement field. Then, the 3D discrete elastic operator defined by finite differences methods is given by

$$A_{el} = D^T D$$

where

$$D = \begin{bmatrix} \sqrt{\mu}\partial_1^h & 0 & 0 \\ \sqrt{\mu}\partial_2^h & 0 & 0 \\ \sqrt{\mu}\partial_3^h & 0 & 0 \\ 0 & \sqrt{\mu}\partial_1^h & 0 \\ 0 & \sqrt{\mu}\partial_2^h & 0 \\ 0 & \sqrt{\mu}\partial_3^h & 0 \\ 0 & 0 & \sqrt{\mu}\partial_1^h \\ 0 & 0 & \sqrt{\mu}\partial_2^h \\ 0 & 0 & \sqrt{\mu}\partial_3^h \\ \sqrt{\lambda+\mu}\partial_1^h & \sqrt{\lambda+\mu}\partial_2^h & \sqrt{\lambda+\mu}\partial_3^h, \end{bmatrix} \tag{B.2}$$

where $\lambda \geq 0$ and $\mu \geq 0$ are lamé parameters, see(**??**).

The Diffusion operator is given by

$$A_{dif} = \nabla = D^T D$$

the Laplacian operator where

$$D = \begin{bmatrix} \partial_1^h & 0 & 0 \\ \partial_2^h & 0 & 0 \\ \partial_3^h & 0 & 0 \\ 0 & \partial_1^h & 0 \\ 0 & \partial_2^h & 0 \\ 0 & \partial_3^h & 0 \\ 0 & 0 & \partial_1^h \\ 0 & 0 & \partial_2^h \\ 0 & 0 & \partial_3^h \end{bmatrix}. \tag{B.3}$$

Note that, one can deduce the diffusion operator from the elastic operator by setting $\mu = -\lambda$.

Appendix C

Riemann-Stieltjes integral and orthogonal polynomials

Consider a finite interval $[a,\, b]$ in the real line and two real functions f and w on $\mathbb{R}$. A subset $P = \{x_0, x_1, \ldots, x_n\}$ of the closed interval $[a,\, b]$ with $a = x_0 < x_1 < \ldots < x_n = b, (n \geq 1)$ is called a partition of $[a,\, b]$. The norm of a partition P is the number

$$V(P) = \max_{1 \leq k \leq max} (x_k - x_{k-1})$$

that is the maximal spacing between two consecutive real points of the partition.

Let us consider two partitions P, Q defined on the interval $[a,\, b]$. If $P \supset Q$, P is said to be finer than Q. One can see that this induces that $V(P) \leq V(Q)$.

A *tagged partition* of $[a,\, b]$ is a pair (P, t) where $P = \{x_0, x_1, \ldots, x_n\}$ is a partition of $[a,\, b]$, and $t = (t_1, t_2, \ldots, t_n)$ is such that $x_{k-1} \leq t_k \leq x_k$.

Now we are ready to define the *Riemann-Stieltjes sum.*

Definition C.1 (*Riemann-Stieltjes (R-S) sum*)

Let (P, t) be a tagged partition of $[a,\, b]$ with $P = \{x_0, x_1, \ldots, x_n\}$. The Riemann-Stieltjes sum of a function f with respect to the density function w is given by

$$s(P, t, f, w) = \sum_{k=1}^{n} f(t_k) \left(w(x_k) - w(x_{k-1})\right). \tag{C.1}$$

The function f is Riemann-Stieltjes integrable with respect to w, if there exists a unique value

$$L = s(p, t, f, w) = \int_a^b f(\lambda) dw(\lambda) \tag{C.2}$$

where w is called the distribution function.

The set $R(w, a, b)$ is the set of Riemann-stieltjes integrable functions with respect to w. If $w(x) = x$, then $R(w, a, b) = R(a, b)$ is the set of Riemann integrable functions on $[a,\, b]$.

The theorem below facilitates the computation of the Riemann-Stieltjes integral.

Theorem C.2

Given a function $f \in R(w, a, b)$, bounded on $[a, b]$ with a continuous derivative w', we have $fw' \in R(w, a, b)$ and

$$\int_a^b f(\lambda) dw(\lambda) = \int_a^b f(\lambda) w'(\lambda) d\lambda.$$

The function $w : [a, b] \Longrightarrow \mathbb{R}$ is said to have *a bounded variation* if

$$V_a^b(w) = \sup \left\{ \sum_{k=1}^{n} |w(x_k) - w(x_{k-1})|, \quad \{x_k\}_{k=0}^{n} \quad \text{a partition of} \quad [a, b] \right\} < \infty.$$

The following theorem ensures sufficient conditions for Riemann-Stieltjes integrability of f.

Theorem C.3

Let f be continuous and w with bounded variations on the interval $[a, b]$. Then $f \in R(w, a, b)$.

A corollary of this theorem states that, if f is continuous and w non decreasing on the interval $[a, b]$, then $f \in R(w, a, b)$. This can be observed by the fact that, any function with bounded variation on $[a, b]$, can be expressed as a difference of two nondecreasing and bounded functions. Applying linearity of the integral one concludes that $f \in R(w, a, b)$. To compute efficiently the R-S integral, on emay define weights at some specific points. The so called *points of increase* are well suited for such points.

Definition C.4 (*Points of increase*)

Given a nondecreasing distribution function w. A point of increase of w is a point in the neighborhood of which the function w is not constant. The finite or infinite number of points of increase of w is denoted $n(w)$.

The points of increase of a distribution function can be exploited for computing the RS integral. Let us consider a distribution function w that is a piecewise constant with N point of increase $\lambda_1, \ldots, \lambda_N$ and positive weights $w_1, \ldots, w_N$ such that

$$w(\lambda) = \begin{cases} 0 & \text{if} \quad a \leq \lambda < \lambda_1, \\ \sum_{j=1}^{i} w_j & \text{if} \quad \lambda_i \leq \lambda < \lambda_{i+1}, \\ \sum_{j=1}^{N} w_j & \text{if} \quad \lambda_N \leq \lambda < b, \end{cases}$$

then the Riemann-Stieltjes integral satisfies

$$\int_a^b f(\lambda) dw(\lambda) = \sum_{i=1}^{N} w_i f(\lambda_i)$$

Definition C.5 (*weighted scalar product*)

Let $\mathcal{P}$ be a space of polynomials and w be a nondecreasing function on $[a, b]$, the

weighted scalar product $\langle .,. \rangle_w : P \times P \Longleftrightarrow \mathbb{R}$ *is defined by*

$$\langle f, g \rangle_w = \int_a^b f(\lambda) g(\lambda) dw(\lambda), \quad \forall\, f, g \in \mathcal{P}. \tag{C.3}$$

It is verified that

$$\begin{cases} \langle f, f \rangle_w & \geq & 0, & \forall\, f, g \in \mathcal{P}, \\ \langle f, f \rangle_w & = & 0, & \text{only if } \ f = 0, \\ \langle f + g, h \rangle_w & = & \langle f, h \rangle_w + \langle g, h \rangle_w, & \end{cases}$$

The norm induced from this scalar product is given by

$$\|f\|_w = \sqrt{\langle f, f \rangle_w} = \sqrt{\int_a^b f(\lambda)^2 dw(\lambda)} \tag{C.4}$$

It is important to note that $\langle .,. \rangle_w$ may not be a scalar product for a general polynomial. For example, when w is a nondecreasing distribution with infinite number of points of increase. However, when w has only N distinct points of increase while $\mathcal{P}_{N-1}$ is a space of polynomials of degree less than N, the mapping $\langle .,. \rangle_w$ is a scalar product on $\mathcal{P}_{N-1}$.

Definition C.6 (*Orthogonal polynomials*)

Let $\langle .,. \rangle_w$ *be a scalar product on* $\mathcal{P}$ *the space of polynomials. A sequence of polynomials* $\phi_0(\lambda), \phi_1(\lambda), \ldots, \phi_N(\lambda), \ldots$ *is* orthogonal *if*

$$\langle \phi_j(\lambda), \phi_k(\lambda) \rangle_w = 0 \quad \forall\, j \neq k,\ j, k = 1 : N,$$

while

$$\langle \phi_j(\lambda), \phi_j(\lambda) \rangle_w \neq 0 \quad \forall\, j.$$

If in addition

$$\langle \phi_j(\lambda), \phi_j(\lambda) \rangle_w = 1 \quad \forall\, j, j = 1 : N,$$

the orthogonal polynomials are called orthonormal. *An orthogonal polynomial is said to be* monic orthogonal *when its leading coefficient is one.*

Theorem C.7 (*[102]*)

Consider a distribution function w *and define the scalar product* $\langle .,. \rangle_w$. *If* w *has infinite points of increase in the interval* $[a, b]$, *then there exists an infinite but unique sequence of monic orthogonal polynomials* $\phi_0(\lambda), \phi_1(\lambda), \ldots, \phi_{N-1}(\lambda), \ldots$. *If* w *has* N *distinct points of increase in* $[a, b]$, *there exists a unique sequence of monic orthogonal polynomials* $\varphi_0(\lambda), \varphi_1(\lambda), \ldots \varphi_N(\lambda)$ *which forms a basis of the space* $\mathcal{P}_{N-1}$ *of polynomials of degree less than* N *[102].*

One can generate the monic orthogonal polynomials so defined by using the Gram-Schmidt method to orthogonalize the sequence of monomials $1, \lambda, \lambda^2, \ldots$ then obtain the sequence of orthonormal polynomials by normalizing these monic orthogonal polynomials. Both monic orthogonal polynomials and orthonormal polynomials have the same roots [102].

Theorem C.8 (*Recurrence relation*)

Let $n(w)$ be the number of points of increase with respect to the distribution function w. Given $n < n(w)$, one gets a sequence of monic orthogonal polynomials $\varphi_0, \varphi_1, \ldots \varphi_n$ by the three-term recurrence as follows:

$$\varphi_{k+1}(\lambda) = (\lambda - \alpha_k)\varphi_k(\lambda) - \delta_k \varphi_{k-1}(\lambda), \quad (k = 0, 1, \ldots, n-1), \tag{C.5}$$

where

$$\begin{aligned} \varphi_{-1}(\lambda) &= 0, \\ \varphi_0(\lambda) &= 1, \\ \alpha_k &= \frac{\langle \lambda\varphi_k, \varphi_k \rangle_w}{\langle \lambda\varphi_k, \varphi_k \rangle_w}, \\ \delta_k &= \frac{\langle \lambda\varphi_k, \varphi_k \rangle_w}{\langle \lambda\varphi_{k-1}, \varphi_{k-1} \rangle_w}, \\ (k &= 0, 1, \ldots, n-1) \end{aligned}$$

Appendix D

Tchebychev preconditioner for multiple right-hand sides

Notice that, all the preconditioners we have discussed until now aim to improve the condition number or the eigenvalues distribution of the matrix A in (4.1). Otherwise, note that the coefficient matrices in linear systems (4.7) and (4.9) remain constant during iterations. Only the right-side changes. Thus, it makes sens to address an efficient resolution of a sequence of linear systems with the same coefficient matrix but multiple right-hand sides. That is to solve

$$Ax_l = b_l, \quad l = 1, 2, \ldots, < \infty. \tag{D.1}$$

This is addressed in the remainder of this chapter following the paper from Golub, Ruiz and Touhami in [103]. In this paper, authors used the Tchebychev polynomial as a filter and preconditioner. This reveals a particular interest for our application since it addresses also the right-hand side that plays an important role on the preservation of the residuals orthogonality for the CG in finite precision as we presented in Section 4.2.1. In addition, authors proposed a way of combining Tchebyshev preconditioner with other preconditioners.

The Tchebyshev polynomial was used to shift a part of the eigenvalue distribution to one, such that the spectrum is more clustered around one. At the same time, this preconditioner is used as a filter of the initial residual $r_0 = b - Ax_0$, such that the eigencomponents in r_0 associated to the shifted eigenvalues are reduced below a threshold ε, called *filter level*. Since the convergence rate of the CG algorithm in finite precision depends on both the initial residual and on the eigenvalue distribution of the coefficient matrix (4.34), this procedure is likely to yield a rapid convergence of the CG algorithm.

However, the efficiency has to be measured by comparing the construction cost of the preconditioner, its application cost and the improvement it yields. For this purpose, the major gain is obtained when some information from a first preconditioned system

$$M^{-1}Ax_1 = M^{-1}b_1, \tag{D.2}$$

is saved and used to solve the following systems (that is $l = 2, \ldots$ in (D.1). This means we combine the Tchebyshev preconditioner with an other preconditioner called first order preconditioner.

According to [103], the gathered information allows the construction of a small dimensional Krylov basis with eigeninformation linked to the smaller eigenvalues (that were not shifted), and this is used to speedup the CG algorithm in further systems.

In practice, the matrix A is factorized as in (4.33) but with two terms

$$A = U\Lambda U^T = U_1\Lambda_1 U_1^T + U_2\Lambda_2 U_2^T, \tag{D.3}$$

where Λ_1 is a diagonal matrix with sorted eigenvalues greater than a *cut-off* parameter $\mu, 0 < \mu < \lambda_{max}$ and Λ_2 is its complementary (i.e $\Lambda = \Lambda_1 + \Lambda_2$).

First, given μ, λ_{max} and a threshold ε, one may define a polynomial of degree m from the Tchebyshev polynomials (4.99) by

$$\tilde{T}_m(\lambda) = \frac{T_m\left(\Psi_\mu(\lambda)\right)}{T_m\left(\Psi_\mu(0)\right)} \tag{D.4}$$

where $\Psi_\mu(\lambda)$ maps the interval $[\mu,\ \lambda_{max}]$ onto $[-1, 1]$. Second, one can fix a degree m of the polynomial $T_m\left(\Psi_\mu(\lambda)\right)$ such that $\dfrac{1}{|T_m\left(\Psi_\mu(0)\right)|} < \varepsilon$ that implies

$$\|\tilde{T}_m(\lambda)\|_\infty < \varepsilon, \ \text{ on } \ [\mu,\ \lambda_{max}]. \tag{D.5}$$

Then, a filtered residual vector z can be determined by applying the polynomial $\tilde{T}_m(\lambda)$ to the residual r_0:

$$z = \tilde{T}_m(A)r_0 = U_1\tilde{T}_m(\Lambda_1)U_1^T r_0 + U_2\tilde{T}_m(\Lambda_2)U_2^T r_0. \tag{D.6}$$

The action of the filter on r_0 can be measured by multiplying both sides of (D.6) and evaluating the norm. This writes

$$\|U_1^T z\|_2 = \|\tilde{T}_m(\Lambda_1)U_1^T r_0\|_2 \leq \|\tilde{T}_m(\Lambda_1)\|_2\|U_1^T r_0\|_2 < \varepsilon\|U_1^T r_0\|_2.$$

Thus, the eigencomponents in r_0 associated to eigenvalues in $[\mu,\ \lambda_{max}$ are reduced bellow the filter level ε on $[\mu,\ \lambda_{max}]$. Note that, since $\tilde{T}_m(\lambda) \leq 1, \forall\ \lambda$ the condition on the filter level can be imposed in (4.101) by imposing $\dfrac{1}{\sigma_m} < \varepsilon$.

The algorithm called *ChebFilter* from [103] is presented in the algorithm D.1 below. Its inputs are respectively, the matrix A, the right-hand side vector b, the cut-off parameter μ, the maximal eigenvalue λ_{max}, the initial solution x_0 and the filter level ε. The outputs are the filtered residual $z = \tilde{T}_m(A)r_0$ and the corresponding solution such that $b - Ax = z$.

Algorithm D.1 (*ChebFilter*)

***Input**: $A, b, x_0, \lambda_{max}, \mu, \varepsilon$*

% outputs are respectively the filtered residual, the current solution
***Output:** $[z,\ x]$*
$\alpha_\mu = \frac{2}{\lambda_{max}-\mu}$; $\quad d_\mu = \frac{\lambda_{max}+\mu}{\lambda_{max}-\mu}$;
$x = x_0$; $\quad z = b - Ax$;
$\sigma_0 = 1,\ \sigma_1 = d_\mu;\ \ k = 1;\ \ y = x;$

$x = x + \frac{\alpha_\mu}{d_\mu} z;$
$z = b - Ax;$
Do while $\frac{1}{\sigma_k} > \epsilon$
$\sigma_{k+1} = 2\sigma_k d_\mu - \sigma_{k-1};$
$p = 2\frac{\sigma_k}{\sigma_{k+1}} (d_\mu x - \alpha_\mu z) - \frac{\sigma_{k-1}}{\sigma_{k+1}} y;$
$y = x$ *and* $x = p;$
$z = r_0 - Ax;$
$k = k + 1;$
End(Do).

Solving the remaining s systems

$$Ax_l = b_l, \quad l = 2, 3, \ldots, s, \tag{D.7}$$

may be facilitated by information from the run of PCG on the first system with the ChebFilter as preconditioner. A Krylov basis W_k is formed by k search direction vectors gathered while running k iterations of the PCG algorithm. From the properties of the CG directions (2.30), this basis is A-orthogonal and thus $A_c = W_k^T A W_k$ is diagonal. This basis is kept for all the systems in (D.7) where r_0 is projected onto AW_k along $ker(W_k^T)$, such that the eigencomponents in a solution corresponding to the smallest eigenvalues are obtained. The pseudo-code of the final algorithm is given in algorithm D.2.

Algorithm D.2 (*[103]*)

1. $[x_1, W] = ChebFilterCG(A, b_1, x_0, \lambda_{max}, tol_1, M_1);$
2. $A_c = W^T A W;$
3. *For* $l = 2 : s$
   (a) $x_0 = W A_c^{-1} W^T b_l;$
   (b) $x_l = PCG(A, b_l, x_0, tol_2, M_1)$
4. *End(For)*

Unfortunately, the tests to combine the Tchebychev polynomial with other preconditioners in MIRP have not yet lead to reliable conclusions. Thus, numerical results are not presented in this thesis.

Another way of improving the optimization process in the MIRP is to approximate better the Hessian of the functional (1.65). This leads to Gauss-Newton strategy. This strategy is presented in general setting in Section 2.3 and particularly for MIRP in Section 1.4.1. In the following section we present only the structure of the matrix obtained to allow comparison with other systems for MIRP.

Appendix E

Flow of diffeomorphisms

Given the manifold $M_d(\Omega)$ The map

$$\Phi : [0,1] \times M_d(\Omega) \to M_d(\Omega) \forall\ t \in [0,1],\ f \in M_d(\Omega)$$

, such that

$$\Phi(t,f) = \Phi_t(f) = f_t,\ \ \Phi_t \in H^1(\Omega)$$

$\Phi_t(f)$ is a curve parametrized by t where $\Phi_0 = Id$ $f_t \in M_d$ is the observed image at time t and $(\Phi_s(f))_{s=0:t}$ is the path followed by $(f_s)_{s=0:t}$. The function $\Phi_t \in H^1$ being differentiable, the velocity field $\dot{\Phi}_t = v_t(\Phi_t)$ is square integrable. Thus, the energy of the curve Φ_t, associated to a metric A in H^1 is

$$E_A(v_t(\Phi_t)) = \int_0^1 \|v_t(\Phi_t)\|_A^2 dt \tag{E.1}$$

By using registration by flow of diffeomorphisms, one aims to determine a transformation Φ_1 such that:

$$\begin{array}{rcl} \Phi_0[I_m(\mathbf{x})] & = & I_m(\mathbf{x}) \\ \Phi_1[I_m(\mathbf{x})] & = & I_f(\mathbf{x}) \\ E_A(v_t(\Phi_t)) & & \text{is minimized} \end{array}$$

Φ_1 is the endpoint of the flow of a time-dependent velocity $v_t(\Phi_t)$ $\Phi_1 = \Phi_0 + \int_0^1 v_t(\Phi_t)$ In these settings, we minimize the functional

$$E(v_t) = \frac{1}{\sigma^2}||I_f[\Phi^{-1}(x)] - I_m(x)||_{L^2}^2 + \int_0^1 ||v_t(\Phi(x))||_{H^1}^2 dt \tag{E.2}$$

where $\sigma > 0$ is a continuous weight parameter .

In this functional the first term minimizes the distance between the two images, the second term minimizes the energy of the curves while enforcing the velocity field to be sufficientely smooth. The required smoothness is enforced by defining a norm on H^1. This is done through a differential operator L such that

$$\|v_t(\Phi_t)\|_{H^1}^2 = \|L(\Phi_t)\|_2 = \|\Phi_t\|_{L^T L}$$

where $L = \alpha\Delta$ or $L = \alpha\nabla$, $\alpha > 0$. Thus, instead of estimating a serie of velocity fields, we need only to estimate an initial velocity field v_0, then this initial velocity plays the role of the instantaneous displacement u. The saugth transformation Φ_1 will evolve from the identity $\Phi_0 = Id$ and follows a dynamical path for a unit time integration Φ_t, $t \in [0\,1]$ via composition. $\Phi_1 = (Id + \frac{\mathbf{v}_{t_T}}{T}) \circ (Id + \frac{\mathbf{v}_{t_{T-1}}}{T}) \circ \ldots \circ (Id + \frac{\mathbf{v}_0}{T})$ This is achieved by interpoling the displacement field in the initial grid and the averaging interpolation command `interpn in Matlab`.

For example, assume we have determined the initial velocity v_0, then consider T a number of time steps

For small deformations one has

$$T = 1$$
$$\Phi_{k+1} = \Phi_k + v_0$$
$$\Phi_{k+1}^{-1} = \Phi_k - v_0$$

although, for large deformations one performs

$$\Phi_{k+\frac{1}{T}} = \Phi_k + (\frac{1}{T})v_0; \;\; \Phi_{k+\frac{2}{T}} = \Phi_{\frac{1}{T}} \circ \Phi_{\frac{1}{T}} \;\; \Phi_{k+\frac{3}{T}} = \Phi_{\frac{1}{T}} \circ \Phi_{\frac{2}{T}}$$

and

$$\Phi_{k+1} = \Phi_{\frac{1}{T}} \circ \Phi_{\frac{T-1}{T}}\,.$$

Appendix F

Numerical experiments (following)

In all these larger images, the FA4DMIR is better since it reduces better the functional value with les cumulated time.

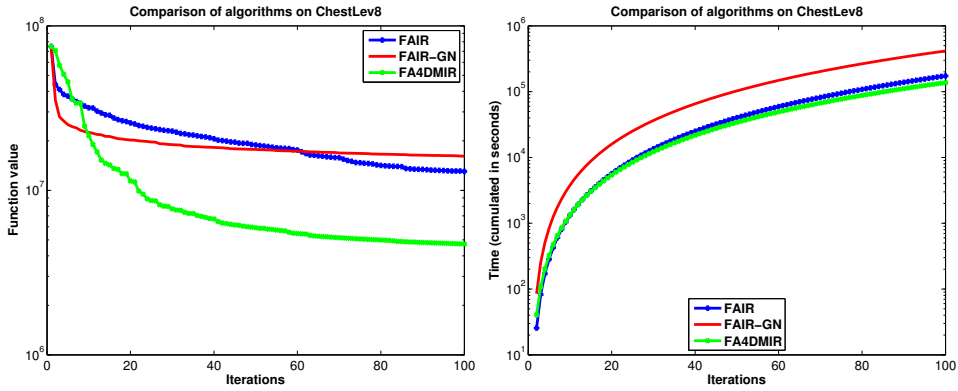


Figure F.1: Chest image level 8: comparison of function value reduction (left) and computing time (right).

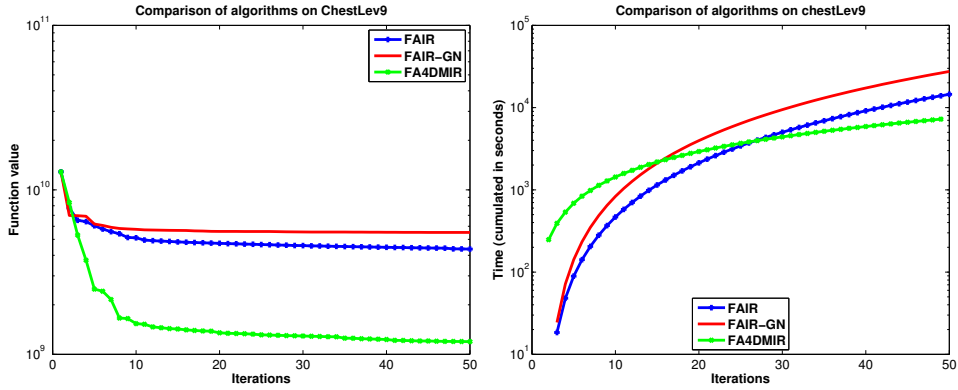


Figure F.2: Chest image level 9: comparison of function value reduction (left) and computing time (right).

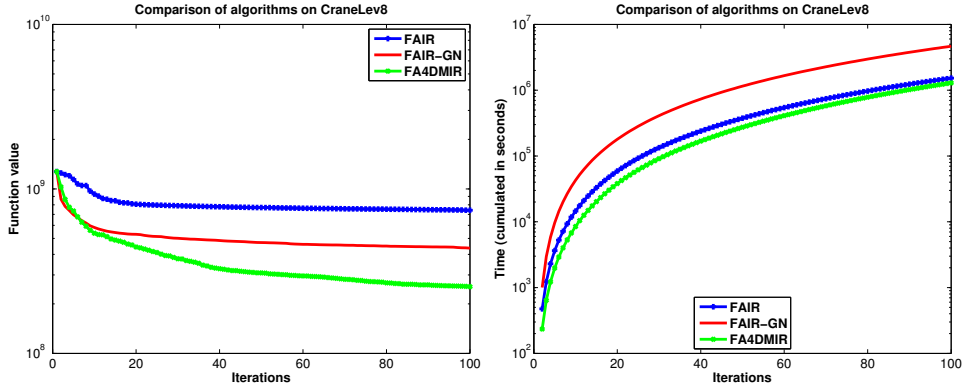


Figure F.3: crane image level 8: comparison of function value reduction (left) and computing time (right).

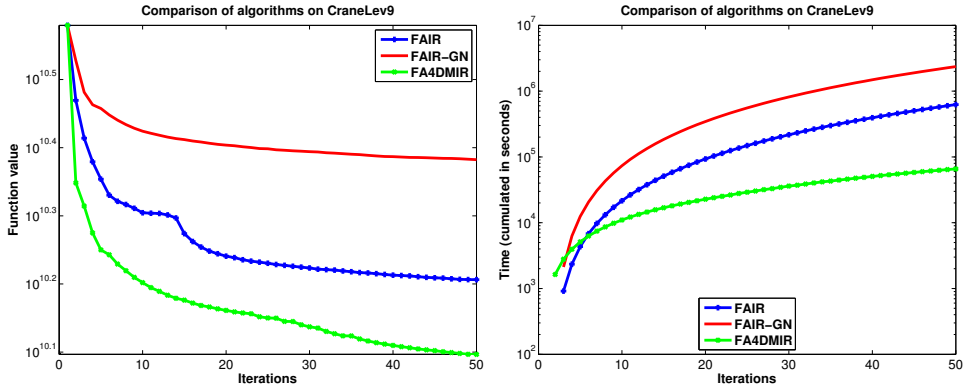


Figure F.4: Crane image level 9: comparison of function value reduction (left) and computing time (right).

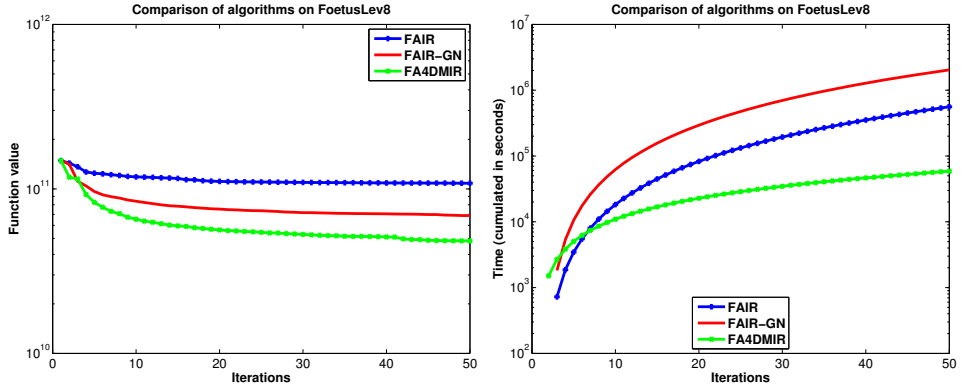


Figure F.5: Foetus image level 8: comparison of function value reduction (left) and computing time (right).

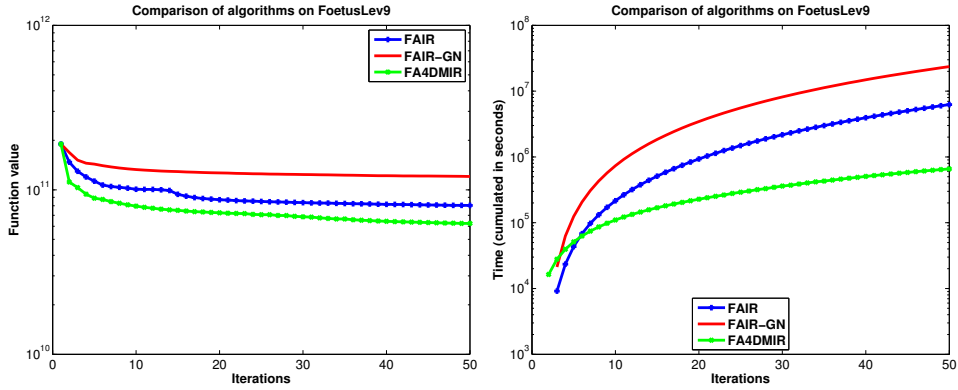


Figure F.6: foetus image level 8: comparison of function value reduction (left) and computing time (right).

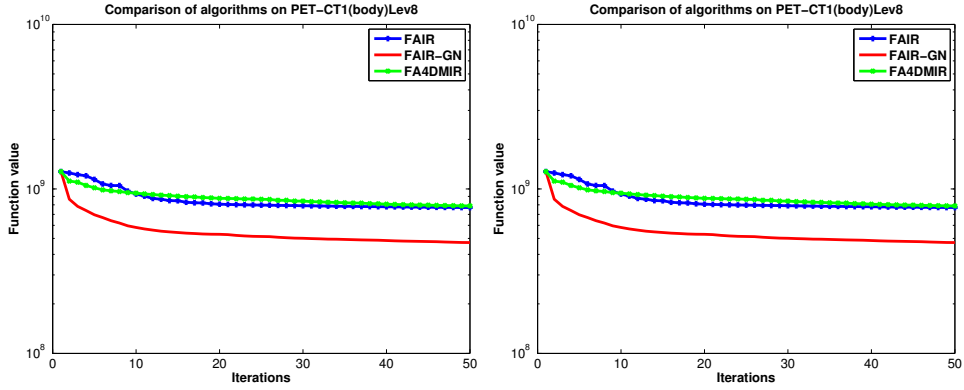


Figure F.7: PET-CT1 image level 8: comparison of function value reduction (left) and computing time (right).

Appendix G

FA4DMIR code organisation

```
addpath /home/justin/2011-08-10-FAIR/
addpath ../configs/
addpath ../drivers/
addpath ../figures/
addpath ../images/
addpath ../miscellaneous/
addpath ../posttreatments/
addpath ../pretreatments/
addpath ../results/
addpath ../solvers/
global Pb;
Pb.solver = 'Tensor';
Pb.file_result = 'resultfile.mat';
Pb.file_in = 'imagesFiles.mat'; % enter the images to be registered.


Pb.parameters = struct('maxIter',n, 'regularizer','mbCurvature','teta',10,      'gri
          'systSol',@myMatrixsolver,'solver','pcgSGS','prereg',0,'beta',10,...
          'tol',1e-6,'sigma',32,'alpha',0.1,'ChebIter',10,'NeumanIter',10,'gn',1,...
          'eigMajorant',20,'build',1,'minLevel',5,'LevMaxC',5,'fType','MatrixMDJ',..
          'systSol',@HT_GenSystSol,'prereg',0,'beta','10',...
          'Prec','InvLaplace','tol',1e-6,'sigma',32,'minLevel',3,'LevmaxC',9,...
          'alpha',0.01,'Nflows',1,'build',1);


switch Pb.solver;

case 'Tensor'
       [yc,I_m,MLHist] = MultLevMIR_HT(imagesFiles,Pb);

case 'Matrix'
```

```
      [yc,I_m,MLHist] = MultLevMIR_MAT(imagesFiles,Pb);
```

In matrix case, only polynomial and LMP preconditioners are added to FAIR package subroutines. Below the code used for tensor case.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%In MultLevMIR_HT(imagesFiles,Pb)%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get multilevel data

 MLT = getMultilevelImage(dataT,paraOpt.omega,struct('minlevel',minLevel));

 MLR = getMultilevelImage(dataR,paraOpt.omega,struct('minlevel',minLevel));

 %minLevel= paraOpt.level;  This means the image with maximal resolution equal 2^(mi

level = minLevel;
paraOpt.level = level;

omega = paraOpt.omega;
% grid type
grid    = paraOpt.grid;

switch grid,
  case 'cellCentered', getGrid = @(m) getCellCenteredGrid(paraOpt.omega,m);
  case 'staggered',     getGrid = @(m) getStaggeredGrid(paraOpt.omega,m);
  case 'nodal',         getGrid = @(m) getNodalGrid(paraOpt.omega,m);
  otherwise, error('nyi');
end;



MLHist = cell(1, paraOpt.nrlevels);



xc      = [];                 % current grid


fprintf('\n\n');
fprintf('%s: MultiLevel Medical Image Registration Htensor\n',mfilename)
fprintf('-- distance=%s, regularizer=%s, alpha=%s, trafo=%s, fileIn=%s\n',...
  distance,regularizer,num2str(regularizer('get','alpha')),trafo, paraOpt.file_in);
```

```
%-------------------------------------------------------------------------
for level=paraOpt.minLevel:paraOpt.LevMaxC


  % save(old grid), update(m,grid,data coefficients)
  xOld  = xc;

  m     = size(MLT{level});

  xc    = getGrid(m);

  paraOpt.m = m;



   % interpolation coefficient

  R= getSplineCoefficients(MLR{level},'dim',paraOpt.dim,'regularizer','gradient','the

  T= getSplineCoefficients(MLT{level},'dim',paraOpt.dim,'regularizer','gradient','the



  if level == minLevel && prereg, % parametric pre-registration


    % for preregistration call Parametric Image Registration from FAIR package ------


  % compute starting guess y0
  if level == minLevel,
    y0 = yRef;  % the best known so far
  else
    % prolongate yc (coarse) y0 (current)
    y0 = xc + mfPu(yc - xOld,omega,m/2);
  end;



  yStop = getGrid(m);
  yc = y0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                     %%%
%%  [yc,I_m,hist] = oneLevelMIR_HT(Rc,T,yc,yStop,paraOpt);%%%
%%                                                                      %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  k = (paraOpt.level-paraOpt.minLevel)+1;
```

```
 MLHist{k} = hist;
 %

  paraOpt.level =  paraOpt.level+1;

end; %For level


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%oneLevelMIR_HT(Rc,T,yc,yStop,paraOpt)%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% handle objective function
 objFunc = @ObjFuncReg;
% initialize the grid
 x0 = getCellCenteredGrid(paraOpt.omega,paraOpt.m);
 Jold = 0;
 % handle large diformation diffeomorphic mapping
 vel2Disp = @velocity2displacement;
% handle system solver by tensor method
 systSol = @HT_LaplSysSolver;

interp = @splineInter;

   tolJ = 1e-6;   tolY = 1e-6;   tolG = 1e-6;


STOP = zeros(5,1);


     [Jc,dJ,Tc] = objFunc(T,Rc,yc,yRef,paraOpt);

      [x, norm_r1] = systSol(dJ,paraOpt);

      Jstop = Jc;

dispHis = @(var)fprintf('%-4d %-8.4e %-8.4e %-8.4e %-8.4e %-8.4e %-4.2e %-2.2e\n',va
hisStr    = {'iter','J','Jold-J','|\nabla J|','|dy|','red','LS','Tcum'};
hist = zeros(paraOpt.maxIter+1,8);

hist(1,:) = [0,Jc,0,norm(dJ),norm(yc-yStop),red,0,0];


dispHis(hist(1,:));

   %[V] = systSol(dJ,paraOpt);

   iter = 1;
  -------------------------------------
```

```
while 1

  yOld = yc;

  STOP(1) = (iter>0) && abs(Jold-Jc)  <= tolJ*(1+abs(Jstop));
  STOP(2) = (iter>0) && (norm(yc-yOld) <= tolY*(1+norm(yStop)));
  STOP(3) = norm(dJ)      <= tolG*(1+abs(Jstop));
  STOP(4) = norm(dJ)      <= 1e6*eps;
  STOP(5) = (iter > paraOpt.maxIter);

  if all(STOP(1:3)) || any(STOP(4:5)), break;  end;
  %if Jc <= 3e7; break;  end;

            V  = HT_GenSystSol(dJ,paraOpt);

           paraOpt.build = 0;

          dy=[reshape(reshape(full(V{1}),paraOpt.m),.[],1);reshape(reshape(full(V{2})


           J0 = Jold;     Jold = Jc;

           % perform Armijo line-search
         [alpha,Yt,LS] =  myRLin(ObjFctn,yc,dy,Jc,J0,dJ);
         if ~LS; break;end


           yc = Yt;
           Jold = Jc;


         [Jc,dJ,Tc] = objFunc(T,Rc,yc,yStop,paraOpt);


  hist(iter+1,:) = [iter,Jc,Jold-Jc,norm(dJ),norm(yc-yOld),red,alpha,Temp];

          dispHis(hist(iter+1,:));

  iter = iter+1;

end
```

Bibliography

[1] A. Sotiras, C. Davatzikos, and N. Paragios, "Deformable medical image registration: A survey," *Medical Imaging, IEEE Transactions on*, vol. 32, no. 7, pp. 1153–1190, 2013.

[2] L. G. Brown, "A survey of image registration techniques," *ACM computing surveys (CSUR)*, vol. 24, no. 4, pp. 325–376, 1992.

[3] S. Saxena and R. K. Singh, "A survey of recent and classical image registration methods," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 7, no. 4, pp. 167–176, 2014.

[4] J.-P. Thirion, "Image matching as a diffusion process: an analogy with maxwell's demons," *Medical image analysis*, vol. 2, no. 3, pp. 243–260, 1998.

[5] M. Lu, "Acceleration method of 3D medical images registration based on compute unified device architecture," *Bio-Medical Materials and Engineering*, vol. 1, no. 24, pp. 1109–1116, 2014.

[6] W. Hackbusch, *Tensor spaces and numerical tensor calculus*, vol. 42. Springer Science & Business Media, 2012.

[7] J. Ballani and L. Grasedyck, "A projection method to solve linear systems in tensor format," *Numerical Linear Algebra with Applications*, vol. 20, no. 1, pp. 27–43, 2013.

[8] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[9] L. Giraldi, *Contributions aux méthodes de calcul basées sur l'approximation de tenseurs et applications en mécanique numérique*. PhD thesis, Ecole Centrale de Nantes, 2012.

[10] J. Modersitzki, *FAIR: flexible algorithms for image registration*, vol. 6. SIAM, 2009.

[11] J. V. Hajnal, D. L. Hill, and D. J. Hawkes, *Medical Image Registration*. The Biomedical Engineering Series, crc press ed., 2001.

[12] S. Angenent, E. Pichon, and A. Tannenbaum, "Mathematical methods in medical image processing," *Bulletin of the American Mathematical Society*, vol. 43, no. 3, pp. 365–396, 2006.

[13] N. Torbati, A. Ayatollahi, and A. Kermani, "An efficient neural network based method for medical image segmentation," *Computers in biology and medicine*, vol. 44, pp. 76–87, 2014.

[14] M. Lysaker, A. Lundervold, and X.-C. Tai, "Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time," *IEEE Transactions on image processing*, vol. 12, no. 12, pp. 1579–1590, 2003.

[15] M. J. McAuliffe, F. M. Lalonde, D. McGarry, W. Gandler, K. Csaky, and B. L. Trus, "Medical image processing, analysis and visualization in clinical research," in *Computer-Based Medical Systems, 2001. CBMS 2001. Proceedings. 14th IEEE Symposium on*, pp. 381–386, IEEE, 2001.

[16] "Plans de direction de l'anatomie de l'homme." `http://data.abuledu.org/wp/?LOM=17334`. Accessed: 2014-06-30.

[17] N. Ayache, "Medical image analysis and simulation," in *Annual Asian Computing Science Conference*, pp. 4–17, Springer, 1997.

[18] F. E.-Z. A. El-Gamal, M. Elmogy, and A. Atwan, "Current trends in medical image registration and fusion," *Egyptian Informatics Journal*, vol. 17, no. 1, pp. 99–124, 2016.

[19] "Apollo medical imaging technology." `https://www.apollomit.com/fusion.htm`. Accessed: 2015-06-30.

[20] "Wikipedia." `https://en.wikipedia.org/wiki/Radon_transform`. Accessed: 2014-06-30.

[21] "Wikipedia." `https://en.wikipedia.org/wiki/Single-photon_emission_computed_tomography`. Accessed: 2017-09-04.

[22] R. K. Hobbie and B. J. Roth, *Intermediate physics for medicine and biology*. Springer Science & Business Media, 2007.

[23] "Open-i Biomedical Image Search Engine." `https://openi.nlm.nih.gov`. Accessed: 2016-09-30.

[24] J. Modersitzki, *Numerical methods for image registration*. Oxford university press, 2003.

[25] "Wikipedia." `https:www.wikipedia.org/wiki/Differentiable_manifold`. Accessed: 2015-06-01.

[26] T. S. Yoo, *Insight into images: principles and practice for segmentation, registration, and image analysis*. AK Peters Ltd, 2004.

[27] B. Fischer and J. Modersitzki, "Ill-posed medicine: an introduction to image registration," *Inverse Problems*, vol. 24, no. 3, p. 034008, 2008.

[28] V. Noblet, *Recalage non rigide d'images cérébrales 3D avec contrainte de conservation de la topologie*. PhD thesis, Université Louis Pasteur-Strasbourg I, 2006.

[29] C. D. Nathan, *Constructing znd solving variational image registration problems*. PhD thesis, Univesity of Oxford, 2009.

[30] J. Schmid, *Knowledge-based deformable models for medical image analysis*. PhD thesis, University of Geneva, 2011.

[31] C. Broit, *Optimal registration of deformed images*. PhD thesis, University of Pensylvannia, 1981.

[32] G. E. Christensen and H. J. Johnson, "Consistent image registration," *Medical Imaging, IEEE Transactions on*, vol. 20, no. 7, pp. 568–582, 2001.

[33] G. E. Christensen, R. D. Rabbitt, and M. I. Miller, "Deformable templates using large deformation kinematics," *Image Processing, IEEE Transactions on*, vol. 5, no. 10, pp. 1435–1447, 1996.

[34] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes, "Computing large deformation metric mappings via geodesic flows of diffeomorphisms," *International journal of computer vision*, vol. 61, no. 2, pp. 139–157, 2005.

[35] S. Marsland and C. J. Twining, "Constructing diffeomorphic representations for the groupwise analysis of nonrigid registrations of medical images," *Medical Imaging, IEEE Transactions on*, vol. 23, no. 8, pp. 1006–1020, 2004.

[36] M. F. Beg and A. Khan, "Symmetric data attachment terms for large deformation image registration," *Medical Imaging, IEEE Transactions on*, vol. 26, no. 9, pp. 1179–1189, 2007.

[37] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Non-parametric diffeomorphic image registration with the demons algorithm," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2007*, pp. 319–326, Springer, 2007.

[38] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Diffeomorphic demons: Efficient non-parametric image registration," *NeuroImage*, vol. 45, no. 1, pp. S61–S72, 2009.

[39] B. T. Yeo, M. R. Sabuncu, T. Vercauteren, N. Ayache, B. Fischl, and P. Golland, "Spherical demons: fast diffeomorphic landmark-free surface registration," *Medical Imaging, IEEE Transactions on*, vol. 29, no. 3, pp. 650–668, 2010.

[40] J. Nocedal and S. J. Wright, *Numerical optimization*, vol. 2. Springer New York, 2006.

[41] R. P. Sampaio, *A trust-region method for constrained derivative-free optimization and worst-case evaluation complexity of non-monotone gradient-related algorithms for unconstrained optimization*. PhD thesis, Univesity of Namur, 2015.

[42] W. F. Trench, "Introduction to real analysis," 2013.

[43] A. R. Conn, N. I. Gould, and P. L. Toint, *Trust region methods*, vol. 1. Siam, 2000.

[44] C. Tannier, *Study of block diagonal preconditionners using partial spectral information to solve linear systems arising in constrained optimization problems*. PhD thesis, Univesity of Namur, 2016.

[45] M. Benzi and M. Tuma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, no. 2-3, pp. 305–340, 1999.

[46] A. Björck, *Numerical methods for least squares problems*. Siam, 1996.

[47] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[48] J. J. Moré and S. M. Wild, "Benchmarking derivative-free optimization algorithms," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 172–191, 2009.

[49] N. Gould and J. Scott, "The state-of-the-art of preconditioners for sparse linear least-squares problems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 43, no. 4, p. 36, 2017.

[50] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab 3.0," *available online, URL: www. tensorlab. net*, 2016.

[51] D. Kressner and C. Tobler, "htucker—a matlab toolbox for tensors in hierarchical tucker format," *Mathicse, EPF Lausanne*, 2012.

[52] I. Oseledets, S. Dolgov, V. Kazeev, O. Lebedeva, and T. Mach, "Tt-toolbox 2.2," *available online, URL: http://spring. inm. ras. ru/osel*, 2012.

[53] S. Klein, M. Staring, and J. P. Pluim, "Evaluation of optimization methods for nonrigid medical image registration using mutual information and b-splines," *Image Processing, IEEE Transactions on*, vol. 16, no. 12, pp. 2879–2890, 2007.

[54] S. Börm, L. Grasedyck, and W. Hackbusch, "Hierarchical matrices," *Lecture notes*, vol. 21, p. 2003, 2003.

[55] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[56] R. H. Chan and M. K. Ng, "Conjugate gradient methods for toeplitz systems," *SIAM review*, vol. 38, no. 3, pp. 427–482, 1996.

[57] P. J. Davis, *Circulant matrices*. American Mathematical Soc., 2012.

[58] M. Jan, "Mathematics meets medecine." presentation, 2008.

[59] T. Gergelits and Z. Strakoš, "Composite convergence bounds based on chebyshev polynomials and finite precision conjugate gradient computations," *Numerical Algorithms*, vol. 65, no. 4, pp. 759–782, 2014.

[60] Y. Notay, "On the convergence rate of the conjugate gradients in presence of rounding errors," *Numerische Mathematik*, vol. 65, no. 1, pp. 301–317, 1993.

[61] A. Greenbaum and Z. Strakoš, "Predicting the behavior of finite precision lanczos and conjugate gradient computations," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 1, pp. 121–137, 1992.

[62] A. Jennings, "Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method," *IMA Journal of Applied Mathematics*, vol. 20, no. 1, pp. 61–72, 1977.

[63] Z. Strakoš and P. Tichý, "On error estimation in the conjugate gradient method and why it works in finite precision computations," *Electron. Trans. Numer. Anal*, vol. 13, no. 56-80, p. 8, 2002.

[64] G. Meurant and Z. Strakoš, "The lanczos and conjugate gradient algorithms in finite precision arithmetic," *Acta Numerica*, vol. 15, pp. 471–542, 2006.

[65] Z. Strakoš, "On the real convergence rate of the conjugate gradient method," *Linear algebra and its applications*, vol. 154, pp. 535–549, 1991.

[66] O. Axelsson and J. Karátson, "On the rate of convergence of the conjugate gradient method for linear operators in hilbert space," 2002.

[67] A. Greenbaum, "The lanczos and conjugate gradient algorithms in finite precision arithmetic," in *Proceedings of the Cornelius Lanczos International Centenary Conference*, pp. 49–60, SIAM, Philadelphia, PA, 1994.

[68] Y. SAAD, *iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

[69] Y. L. B. Eng, *The use of parallel polynomial preconditioners in the solution of systems of linear equations*. PhD thesis, University of Ulster, 2005.

[70] Q. Alfio, S. Riccardo, and S. Fausto, *Méthodes numériques. Algorithmes, analyse et applications*. Springer, 2007.

[71] J. Tshimanga Ilunga, *On a class of limitted memory preconditionners for large-scale nonlinear least-squares problems (with application to variational acean data assimilation)*. PhD thesis, Univerité de Namur, 2007.

[72] S. Gratton, A. Sartenaer, and J. Tshimanga, "On a class of limited memory preconditioners for large scale linear systems with multiple right-hand sides," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 912–935, 2011.

[73] O. Axelsson, "A survey of preconditioned iterative methods for linear systems of algebraic equations," *BIT Numerical Mathematics*, vol. 25, no. 1, pp. 165–187, 1985.

[74] S. Demko, W. F. Moss, and P. W. Smith, "Decay rates for inverses of band matrices," *Mathematics of computation*, vol. 43, no. 168, pp. 491–499, 1984.

[75] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[76] K. Chen, *Matrix preconditioning techniques and applications*, vol. 19. Cambridge University Press, 2005.

[77] J. Chen and Y. Saad, "On the tensor svd and the optimal low rank orthogonal approximation of tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 4, pp. 1709–1734, 2009.

[78] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[79] C. Jud, M. Lüthi, T. Albrecht, S. Schönborn, and T. Vetter, "Variational image registration using inhomogeneous regularization," *Journal of mathematical imaging and vision*, vol. 50, no. 3, pp. 246–260, 2014.

[80] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[81] V. A. Kazeev and B. N. Khoromskij, "Low-rank explicit qtt representation of the laplace operator and its inverse," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 3, pp. 742–758, 2012.

[82] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 71–79, 2014.

[83] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use svd in many dimensions," *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3744–3759, 2009.

[84] G. Beylkin and M. J. Mohlenkamp, "Algorithms for numerical analysis in high dimensions," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 2133–2159, 2005.

[85] P. Comon, "Tensors: a brief introduction," *IEEE Signal Processing Magazine*, vol. 31, no. 3, pp. 44–53, 2014.

[86] L. Grasedyck, *Polynomial approximation in hierarchical Tucker format by vector-tensorization*. Inst. für Geometrie und Praktische Mathematik, 2010.

[87] S. Aja-Fernandez, R. de Luis Garcia, D. Tao, and L. Xuelong, *Tensors in image processing and computer vision*. Springer, 2009.

[88] L. De Lathauwer, B. De Moor, and J. Vandewalle, "On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[89] S.Sauter, "Numerical tensor calculus." Cours, 2013.

[90] L. D. Lathauwer, "Tensor decompositions and blind signal separation." Winter school, KU Leuven, 15-16 january, 2016.

[91] N. Lee and A. Cichocki, "Regularized computation of approximate pseudoinverse of large matrices using low-rank tensor train decompositions," *SIAM Journal on Matrix Analysis and Applications*, vol. 37, no. 2, pp. 598–623, 2016.

[92] I. Oseledets, E. Tyrtyshnikov, and N. Zamarashkin, "Tensor-train ranks for matrices and their inverses," *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.*, vol. 11, no. 3, pp. 394–403, 2011.

[93] I. Oseledets and E. Tyrtyshnikov, "Tt-cross approximation for multidimensional arrays," *Linear Algebra and its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[94] I. V. Oseledets and S. Dolgov, "Solution of linear systems and matrix inversion in the tt-format," *SIAM Journal on Scientific Computing*, vol. 34, no. 5, pp. A2718–A2739, 2012.

[95] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A683–A713, 2012.

[96] B. N. Khoromskij, I. V. Oseledets, *et al.*, "Dmrg+ qtt approach to computation of the ground state for the molecular schrödinger operator," 2010.

[97] S. R. White, "Density-matrix algorithms for quantum renormalization groups," *Physical Review B*, vol. 48, no. 14, p. 10345, 1993.

[98] L. Grasedyck, "Existence and computation of low kronecker-rank approximations for large linear systems of tensor product structure," *Computing*, vol. 72, no. 3, pp. 247–265, 2004.

[99] I. Oseledets, "Numerical tensor methods and applications." Lectures, 2013.

[100] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Approximate iterations for structured matrices," *Numerische Mathematik*, vol. 109, no. 3, pp. 365–383, 2008.

[101] I. V. Oseledets, D. V. Savostyanov, and E. E. Tyrtyshnikov, "Linear algebra for tensor problems," *Computing*, vol. 85, no. 3, pp. 169–188, 2009.

[102] T. Gergelits and T. Gergelits, "Krylov subspace methods: Theory, applications and interconnections," *rem*, vol. 4, p. 20.

[103] G. H. Golub, D. Ruiz, and A. Touhami, "A hybrid approach combining chebyshev filter and conjugate gradient for solving linear systems with multiple right-hand sides," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 3, pp. 774–795, 2007.